SOFAT

A scenario Oracle and Formal Analysis Toolbox

Version 3.0.0

September 2008

Table of Contents

Introduction

SOFAT is the acronym for Scenario Oracle and Formal Analysis Toolbox. As this name suggests it is a formal analysis toolbox for scenarios. Scenarios are informal descriptions of behaviors of distributed systems. SOFAT allows the edition and analysis of distributed systems specifications described using Message Sequence Charts, a scenario language standardized by the ITU [Z.120]. The main functionalities proposed by SOFAT are the textual edition of Message Sequence Charts, their graphical visualization, the analysis of their formal properties, and their simulation. The analysis of the formal properties of a Message Sequence Chart specification determines if a description is regular, local choice, or globally cooperative. Satisfaction of these properties allow respectively for model-checking of logical formulae in temporal logic, implementation, or comparison of specifications. All these applications are either undecidable problems or unfeasible if the Message Sequence Chart description does not satisfy the corresponding property. The SOFAT toolbox implements most of the theoretical results obtained on Message Sequence Charts this last decade. It is regularly updated and re-distributed.

The purpose of this software is twofold:

- Provide a scenario based specification tool for developers of distributed applications

- Serve as a platform for theoretical results on scenarios and partial orders

This document is the User Manual for SOFAT. As any user manual, it details the main features of the software. However, we believe that a basic comprehension of scenarios and of properties of scenario languages is essential to benefit fully from the functionalities of SOFAT.

This document is then divided into three parts:

- Part1 : Message Sequence Charts : this part of the document describes the syntax of the scenario language used by SOFAT.

- Part 2 : Properties of Message Sequenc Charts : this part of the document makes an inventory of some properties of scenarios.

- Part3 : describes the main features of SOFAT, and the architecture of the software.

Part 1

MESSAGE SEQUENCE CHARTS

1 Message Sequence Charts

Message Sequence Charts (or MSC) is a requirement language standardized by the International Telecommunication Union [Z120]. Its role is to represent behaviors of distributed systems. These behaviors can be seen as specifications, as collections of output traces, etc. No indication is given by ITU on the expected role of the language.

MSCs depict the behaviors of entities (frequently called **instances**) that communicate via asynchronous messages. The nature of instances is not precised, and can range from processes, to human beings. MSCs are defined by two levels of specification, namely basic Message Sequence Charts (or **bMSCs**) and High-level Messages Sequence Charts (or **HMSCs**). An MSC description is given by several bMSCs and several HMSCs enclosed into an **MSC document**. bMSCs and HMSCs possess both a graphical and textual representation. In this documentation, we will show both formalisms to illustrate the examples and functionalities of SOFAT. The textual and graphical forms for MSCs are part of a standard maintained by the ITU [Z.120]. SOFAT reads MSC documents in textual form. It can also display HMSCs and bMSCs in graphical form. So far, SOFAT provides no means to edit MSCs or HMSCs in graphical form.

1.1 Basic MSCs

A basic MSC defines a simple scenario, i.e. an abstraction of a system behaviour. Within bMSCs, processes are called instances, and are represented by a vertical axis, along which events are put in top down order. Message exchanges are represented by arrows labeled by message names from the emitting to the receiving instance. No assumption whatsoever is made about the communication medium.
The example below is a simple basic msc called ACCEPT, that depicts a message exchange.

```
msc Accept;

    instance A;

        in conf from NB;

    endinstance;

    instance NB;

      out conf to A;

    endinstance;

endmsc;
```
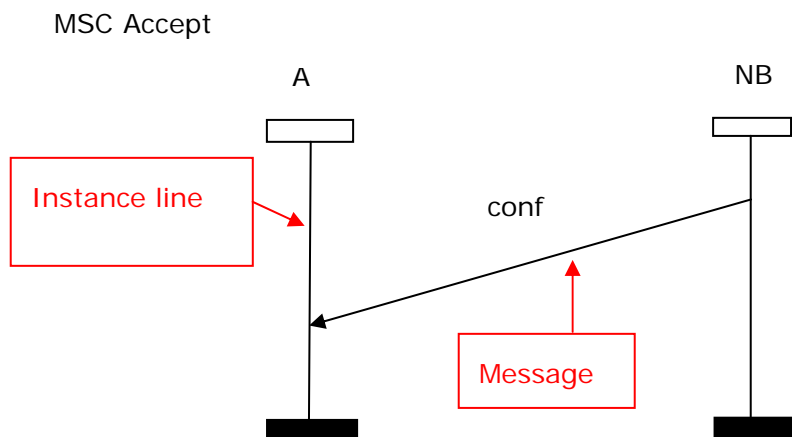
1.1.1 Graphical Representation

MSC Accept

Figure 1: graphical representation for basic MSCs

The drawing in Figure xxx is the graphical representation for the MSC ACCEPT in previous example. Instances are symbolized by vertical lines, messages with arrows.

In this example, A and NB are instances. Both instances exchange a message of type "conf". A and NB can be processes, machines, human beings. Their nature is not specified. Message "conf" is supposed asynchronous, that is the transmission of the message is not immediate, and there is some time elapsing between sending and reception of a message.

In their textual form, basic MSCs read by SOFAT are defined instance by instance[1]. A basic MSC definition starts with : *msc <msc name>;* , and ends with *<endmsc>;*, and contains a list of instance definitions between the starts and end of the basic MSC. Each instance definition starts with instance *<instance name>* ; and ends with *endinstance*, where *<instancename>* is the chosen name for the defined instance. The definition then contains a list of event statements located between the start and the end of the instance.

An event statement can be a message sending, a message reception, an atomic action, or a timer operation. Each definition of an event ends with a semicolon. We give below the syntax of these events:

Message sending : *out <message name> to <instance name>;*

Meaning : currently defined instance sends an asynchronous message of type <message name> to instance <instance name>

---

[1] Note that there exists a different representation (called "event driven" representation) that lists all events one after another. However, this representation is less readable, and we have chosen to stick to an "instance driven" representation of basic MSCs.

Message reception *: in <message name> from <instance name>;*

Meaning : currently defined instance receives a message of type *<message name>* from instance *<instance name>*

Atomic action : *action <action name> ;*

Meaning : currently defined instance executes an atomic action called *<action name>*. An atomic action is a local operation that does not involve a communication with other instances. This can be for instance the computation of a value, a storage in a local database, etc.

In addition to communications and communications, basic MSCS allow for the description of timers. Timers can be set to a given value, or reset. After a certain amount of time has elapsed, timers expire, producing a timeout.

Timer setting : *set <timername>, <value> <unit>;*

Meaning : the timer *<timername>* is set, and should expire within *<value>* units of time. (milliseconds, seconds, unspecified time units, …)

Timer resetting : *reset <timer name>*

Meaning : the timer *<timer name>* that was previously set is reset.

Timeout : *timeout <timername>*

Meaning : the timer <timername> has expired.

1.1.2 Z120 features not addressed :

Several other constructs are proposed in the Z.120 standard, but not all of them are currently supported by SOFAT.

Conditions : The semantics of conditions does not reach a general agreement. For some authors, conditions are only labels that describe a possible state of the distributed system ("IDDLE"). For some others, conditions are guards, and may contain a Boolean predicate, evaluated at execution time.

Lost and found messages : Z.120 allows for the definition of lost and found messages, that is message that are sent but never received by the destination instance, or messages for which sending was not defined, but that nevertheless reach an instance. SOFAT does not support these features for the moment. Note however that lost and found messages can be described by atomic actions.

Gates : Z.120 allows for the emission of messages to gates instead of instances. This feature is not supported for the moment.

MSC AllFeatures

A   B

conf

act

T = d1 ms

T

T = d2 ms

req

T
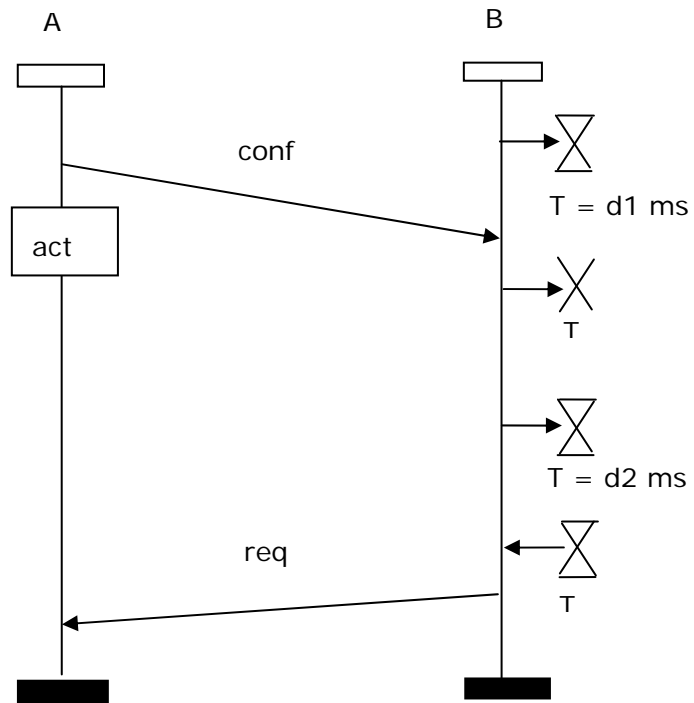
Figure 2: A basic MSC with all MSC features

Figure 2 represents a basic MSC with all the features supported by SOFAT. The textual representation for this example is given below. One may note from this example that comments are also allowed.

Msc Allfeatures;

/* This description contains all features of SOFAT */

Instance A;

   out conf to B ;

   action act ;

   in req from B;

Endinstance;

Instance B;

   set T d1 ms;

   in conf from A ;

   reset T;

   set T d2 ms;

   timeout T;

   out req to A ;

eninstance ;

endmsc;

Note that no time analysis is performed by SOFAT for the moment. Hence, no quantitative analysis is performed for scenarios that contain timers, and inconsistent values can be given to timers.

1.1.3 Basic MSc semantics:

MSCs can be seen as partial orders. All events defined in a bMSC are ordered along their instance, and all message sendings precede the corresponding reception.

The interleaved semantics of a basic MSCs is a collection of event sequences. An event can be executed if and only if all preceding events have been executed. We do not detail here the semantics, and refer interested readers to [AnnexB]. On the example of Figure 3 next page, the interleaved semantics is the sequence

User!Sys(Interrupt).Sys?User(Interrupt).Sys!DB(Stop).DB?Sys(Stop).DB!User(Ack).User?DB(Ack)

where P!Q(m) denotes the sending of message m from P to Q and Q?P(m) its reception.
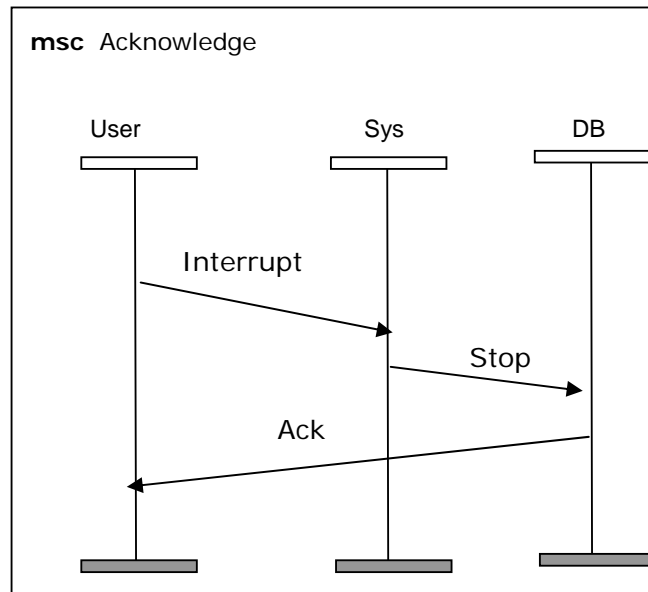
Figure 3 : An example basic MSC

1.2 High level MSCs

Basic MSCs only describe rather simple and linear behaviors of distributed systems. High level msc allows for the definition of more elaborated scenarios by combination of basic mscs and High-level MSCs.

The main constructs used by High-level MSCs are references to basic MSCs of High-level MSCs, sequence, choices.

The syntax of High-level MSC textual descriptions is as follows :

msc <msc name>;

expr <line name>;

{line definition}[*]

endmsc;

expr <line name> defines on which line of the description the HMSC description begins. Each line of the HMSC starts with a line name, and is followed by an HMSC statement. It is then of the form

<line name> : <MSC reference> | connect seq (<line name>);

<line name> : <MSC reference> | connect seq (<line name> alt ….alt <line name>);

<line name> : end;

With this syntax, High-level MSCs describe automata that are labeled by references to basic MSCs or to other HMSCs. The example below is a simple example that describes a choice between two possible behaviors that start at line L2 and L4. Note that this HMSC contains a loop, as it is possible to get back to line L1.

```
msc one ;

  expr L1;

  L1: connect seq (L2 alt L4) ;
  L2: M1 seq (L3);

  L3: H1 seq (L1);

  L4: M2 seq (L5);

  L5: end;

 endmsc;
```

The graphical representation of this HMSC is given in Figure 4 next page.

1.2.1 Graphical Representation

HMSC also have a graphical representation that resembles automata. Beginning of the specifications are represented with a downward triangle, end of specifications by an upward triangle. References to basic MSCs or HMSCs are represetned with rectangles labeled by the name of the referred MSC, connectors by circles.

The graphical representation of the High level MSC one in previous example is depicted in Figure 4 below.

MSC ONE
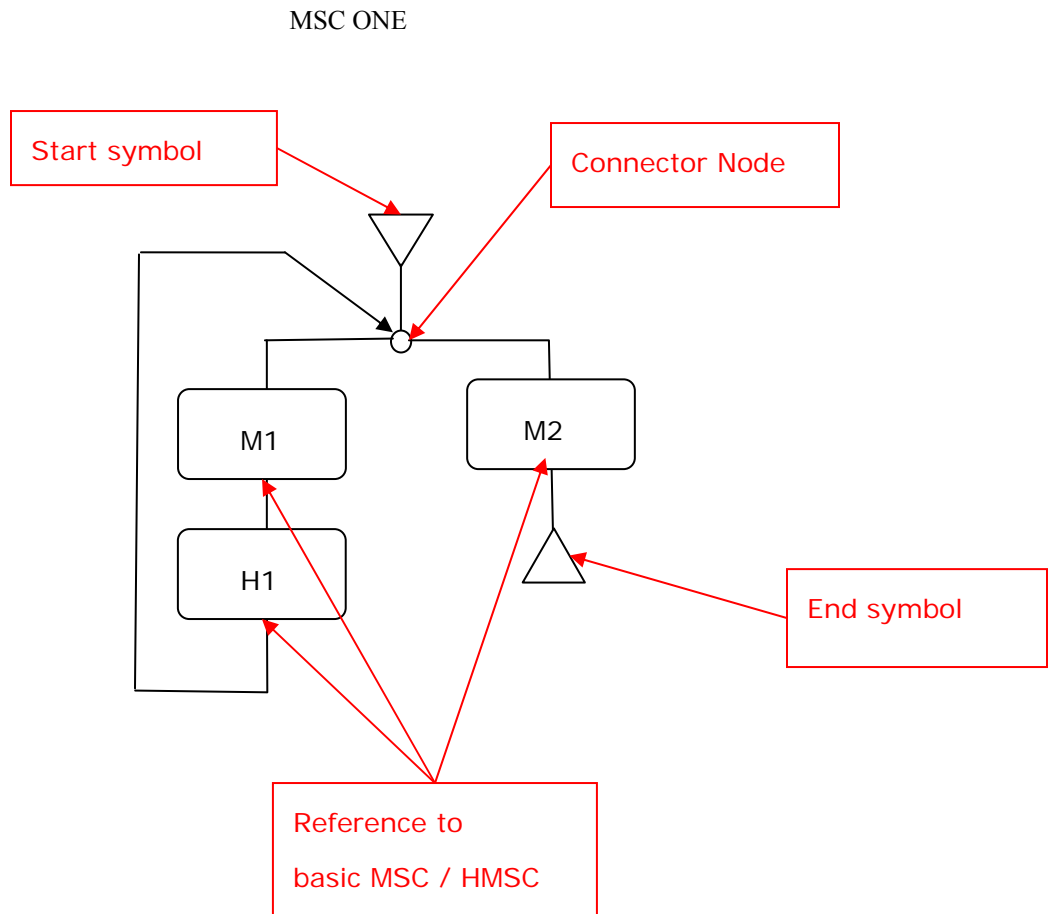
Figure 4 Graphical representation for High-level MSCs

1.2.2 High-level MSCs hierarchy

As High-level MSCs can also reference HMSCs, an MSC document defines an HMSC hierarchy. This hierarchy can be flattened to obtain a HMSC that only references basic MSCs. Hence the two HMSCs below are equivalent, and can be obtained by replacement of the reference to HMSC "two" in msc "one" by a sequence of nodes described in MSC "two". Before studying MSC specifications, SOFAT flattens the HMSC hierarchy. This flattened HMSC can then be saved.

Mscdocument hierarchy;

msc one;

expr L1;

L1 : two seq (L2);

L2 : M1 seq (L3);

L3 : end;

endmsc;

msc two;

expr X1;

X1 : M2 seq (X2);

X2 : M3 seq (X3);

X3 : end;

endmsc;

endmscdocument;

Mscdocument flattened;

msc flat_one;

expr L1;

L1 : M2 seq (L2);

L2 : M3 seq (L3) ;

L3 : M1 seq (L4);

L4 : end;

endmsc;

endmscdocument;

Two equivalent MSC documents.

Hierarchy and referencing can be the source for several problems:

- Some basic/high-level MScs that are referenced are not defined in the msc document.

- The hierarchy is cyclic : a HMSC H1 references a HMSC H2 that references a HMSC H3 ….that references a HMSC Hn that references H1. Clearly, cyclic hierarchies can not be flattened.

SOFAT detects missing references and cyclic hierarchies. When a HMSC description is parsed and analyzed, if the hierarchy is not cyclic, then the description is flattened, and all formal manipulations are then performed on the flattened hierarchy.

1.2.3 A note on MSC sequence

High-level MSC is a mechanism that allows for the composition of basic MSCs. They can be seen as automata labeled by basic MSCs. The behaviors described by a HMSC are then the sequences of bMSCs along the path of the HMSC description. The sequence of two MSCs does not impose any synchronization between the two BMSCs : a part of the behavior described in the second MSC can start even if the behavior described in the first MSC is not completely achieved.

The interpretation of sequential composition of MSCs is an instance by instance merging of two basic MSCs. The following descriptions are then equivalent :

```
mscdocument sequence;                        mscdocument composed;

msc h1;

    Expr l1;                                 msc comp;

    L1 : M1 seq (L2);                            instance User;

    L2 : M2 seq (L3) ;                               out connect to Sys;

endmsc;                                              in confirm from Sys;

msc M1;                                          endinstance;

    instance User;                           Instance sys;

        out connect to Sys;                      in connect from User;

    endinstance;                                 out confirm to User;

    instance Sys;

        in connect from User;                    endinstance;

    endinstance;                             endmsc;

endmsc;                                      endmscdocument;

msc M;

    instance User;

        in confirm from Sys;

    endinstance;

    instance Sys;

        out confirm to User;

    endinstance;

endmsc

endmscdocument;
```

Two equivalent MSC documents

1.2.4 A note on Atoms

As Mscs can be composed sequentially, a question that naturally arises is whether a basic MSC is the product of the composition of several smaller basic MSCs. The smallest non-empty MSCs are called Atoms.

[Helouet00] has proposed an algorithms to decompose a MSC into an equivalent sequence of basic MSCs. Similarly a HMSC can be transformed into an equivalent HMSC labeled only with atoms. The Figure 5 below shows an example of non atomic basic MSC, and enclosed in dotted lines, the atoms it contains.
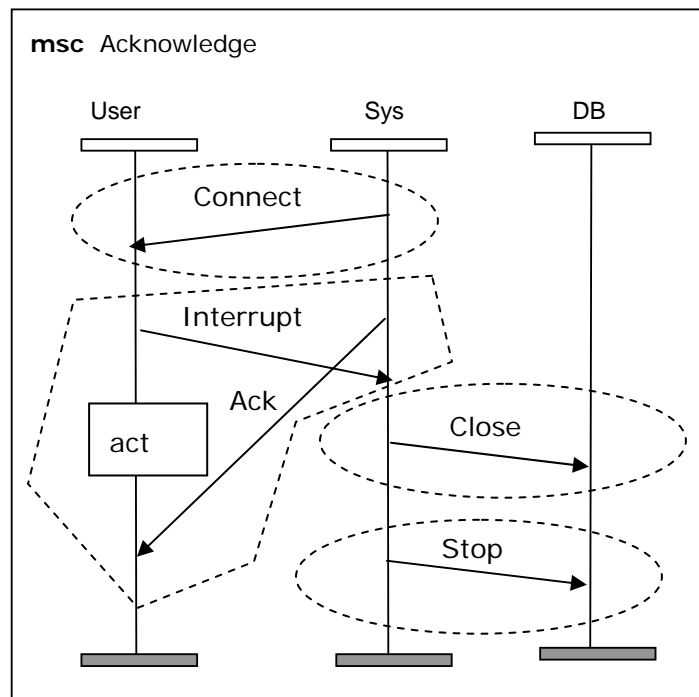
Figure 5 : A basic MSC and its atoms

1.2.5 High level MSCs semantics

For Message Sequence Charts, the runs of a description are defined by the semantics provided by appendix B to Z.120 [Z120 AnnexB]. Informally, the semantics of an HMSC can be interpreted as the set of all basic MSCs it generates, or as the set of all linearizations of the basic MSCs it generates. For a given MSC description M, we will denote by $L(M)$ the set of all runs defined by M. We will also denote by F(M) the set of basic MSCs (bMSCs) described by an MSC description M. Note also that an MSC description does not only represent a set of runs, but also a set of basic MSCs, that can be obtained by unfolding loops, replacing alternatives by a single choice, etc. Consider, for instance the MSC description of Figure 6. This description defines two possible MSCs (Alternative_OK and Alternative_Late) that are depicted in Figure 7.
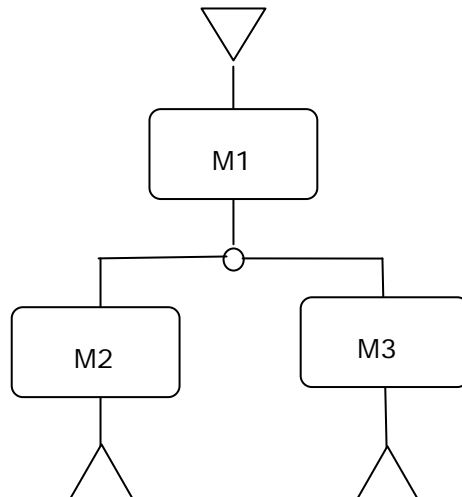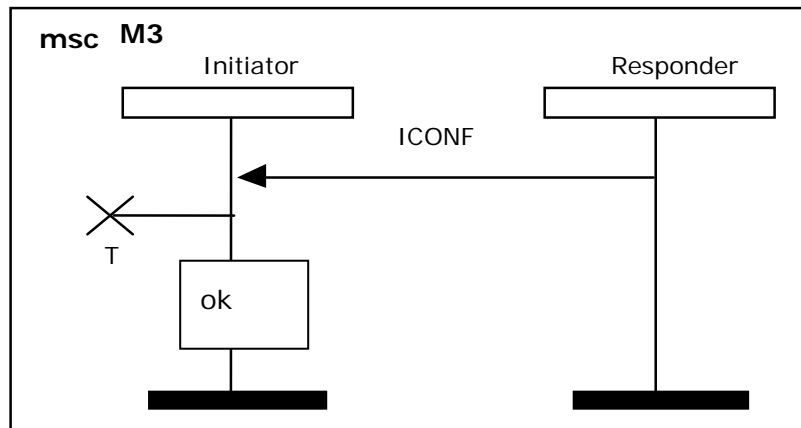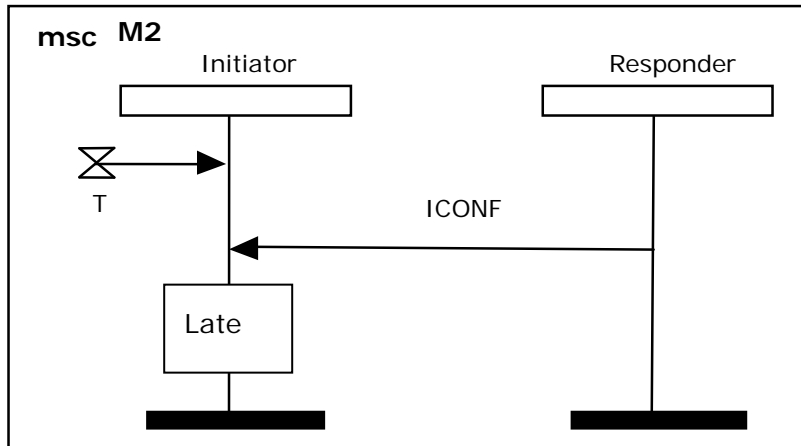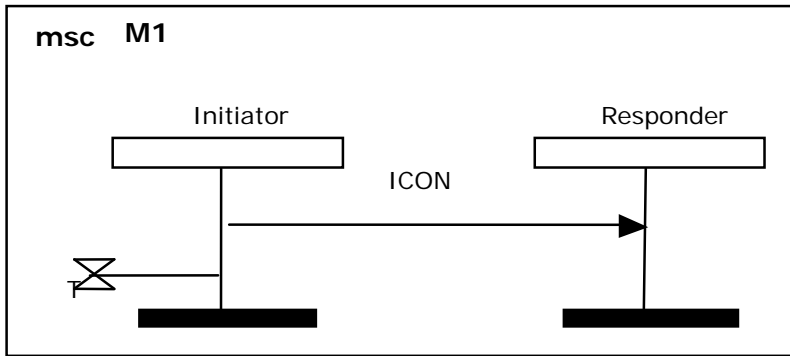
Figure 6 An MSC description with an alternative
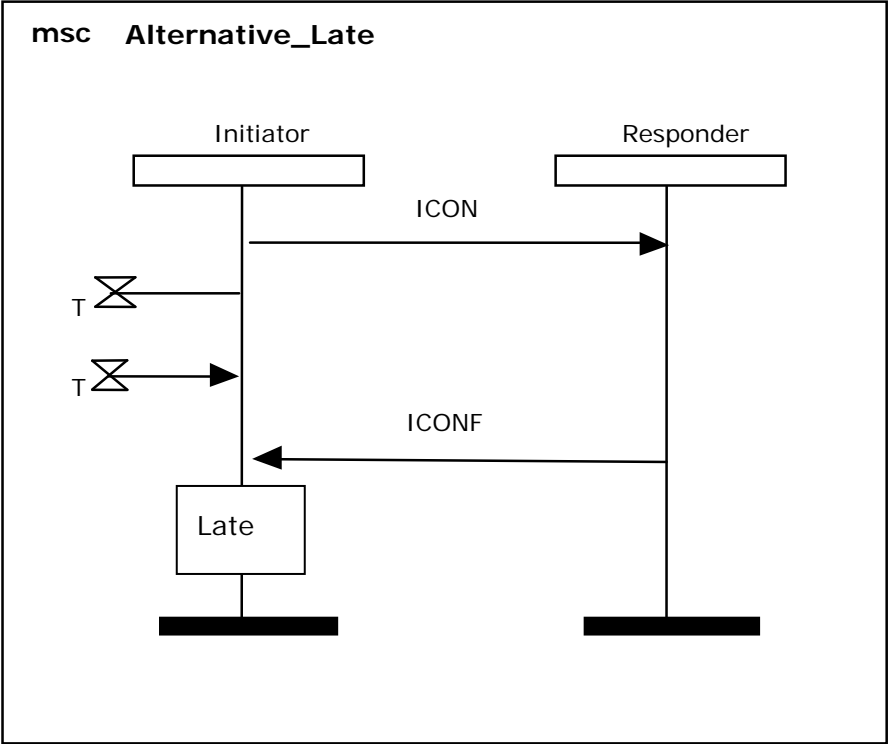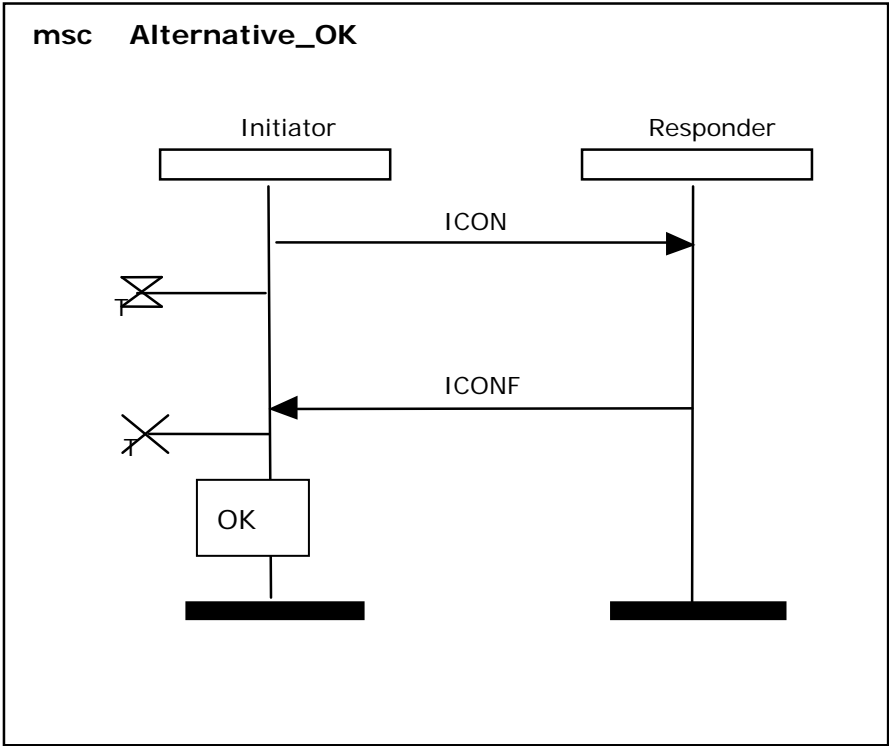
Figure 7 basic MSC interpretation of the MSC in Figure 6

1.3 MSC Documents

MSC documents collect basic MSCs and High-level MSCs into a single description. They have no graphical representation. The syntax of an MSC document is as follows:

Mscdocument *<documentname>;*

<msc list >

Endmscdocument;

Where *<documentname>* is the name of the described msc document, and *<msc list>* is a list of basic mscs or HMSCs.

Part 2

Properties of Message Sequence Charts

2 Properties of Message Sequence Charts

2.1 Frequently addressed problems with MSCs

This part of the documentation addresses the problem of possible applications of Message Sequence Charts (MSCs). The Z.120 standard states that MSCs are meant to "describe interactions between a number of independent message passing instances". In addition to this, MSCs are "a scenario language", "a graphical language", "a formal language", and are "widely applicable", that is not "tailored for one single application domain". These application domains are not explicitly defined in the Z.120 standard, which leaves ground for the following interpretation that MSCs can be used in almost any context, without restriction. This is not the case, and recent literature has shown that with their whole expressive power, several applications of MSCs were impracticable.

Among the applications of MSC, the following are frequently addressed:

- Model checking,

- Comparison of specifications,

- Specification and implementation.

This list is not exhaustive, but has been well covered by the literature in the last decade. In particular, several publications have shown that these applications can be undecidable problems for MSCs, that is there exists no algorithm that takes as an input any MSC and that terminates with as output a correct solution. We give below a non exhaustive list of known decision problems that are impracticable in general for MSCs, and a list of syntactic criteria that ensure decidability of some of the problems listed. The property analysis of SOFAT identifies whether a MSC document matches these criteria.

2.1.1 Model checking

The usual definition of model checking is verifying whether a logic formula ϕ, described with a specific syntax, is satisfied by a model M. This is written $M \models \phi$. Several popular logics exist. We can cite Linear Temporal Logic (LTL), Computational Tree Logic (CTL), CTL*, Alternating-time Temporal Logic (ATL), and the modal μ-calculus. For an introduction to model checking and logics, interested readers may consult [Clarke99, Holzmann99].

Temporal logics are frequently used to ensure that modelled systems satisfy some safety or liveness properties. Logics can address properties of global states, or question the structure of the model itself, and the interpretation of a formula ϕ depends on the semantics of the logic. Similarly, the usual interpretation of model checking is that the formulae address properties of runs and global states of the model.

An example of Linear Temporal Logic (LTL) formula is:

$$\phi_{1} = \mathbf{G}(a \Rightarrow \mathbf{F}\ b)$$

With the LTL semantics, if *a* and *b* are action names, this property means that it is always true that when action *a* is played, then action *b* is eventually played. A whole description of LTL and other logics is beyond the scope of this document.

Frequently, checking a logical formula over runs of a model *M* is equivalent to verifying joint properties of runs of the model and of a finite state automaton R_ϕ computed from the formula. For instance, an automaton $R_{\neg 1}$ associated to the negation of formula ϕ_1 above is depicted in Figure 3. This automaton describes all runs that do not satisfy $_1$. Checking whether a MSC M satisfies ϕ_1 consists in verifying that the set of runs described by M and by the automaton $R_{\neg 1}$ are disjoint.
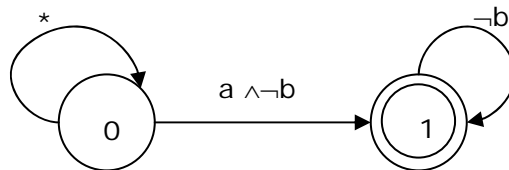

Figure 8 An automaton $R_{\neg\phi1}$ collecting runs that satisfy $\neg\phi1$

To model-check logical properties on Message Sequence Charts specifications, tools have to provide an answer to a question of the kind:

- *(Mc1) L(M)* $\subseteq R_\phi$?

- *(Mc2) R_\phi \subseteq L(M)* ?

- *(Mc3) L(M)* $\cap R_\phi = \varnothing$?

Where M is the MSC description, ϕ a logical formula R_ϕ a finite state automaton that describes sets of runs that satisfy (or do not satisfy) $\tilde{\phi}$. Here, *Mc1* occurs when R_ϕ models all acceptable behaviors: the behaviors of the MSC specification should be contained in the behaviors of R_ϕ and the user wants a positive answer. *Mc2* occurs when R_ϕ models the bad properties of all behaviors that should not occur, and the user expects a negative answer. *Mc3* occurs when R_ϕ models behaviors that have some undesired property, and the user expects a negative answer. Within this setting, comparison of the runs an MSC with a finite state automaton should be a tractable problem.

2.1.2 Comparison of MSC descriptions

Comparison of MSC descriptions is another problem close to model checking. As there might be several ways to describe similar behaviors with Message Sequence Charts, the questions of the equivalence of two descriptions might be interesting. For two MSC descriptions *M1* and *M2*, one may also want to verify that *M2* is an extension of *M1*, i.e. that all behaviors described by *M1* can be found in *M2*. This comparison can occur at the level of runs, or at the level of basic MSCs generated by two MSC specifications. Hence, comparing two MSC specifications M1 and M2 resumes to answering any of the six following questions:

- *(EquivL)* $L(M1) = L(M2)$?

- *(RefL)* $L(M1) \subseteq L(M2)$?

- *(IntL)* $L(M1) \cap L(M2) = \varnothing$?

- *(EquivF)* $F(M1) = F(M2)$?

- *(RefF)* $F(M1) \subseteq F(M2)$?

- *(IntF)* $F(M1) \cap F(M2) = \varnothing$?

2.1.3 Specification and implementation

Message Sequence Charts allow for the description of interactions. It is then tempting to consider them as a specification or even as a development and programming language. However, not all MSC descriptions can be implemented. Consider for instance the example of Figure 14. In this MSC, two systems are performing actions, namely *count* for instance *System1*, and *recount* for instance *System2*. Implicitly, in all bMSCs depicted by this description, the number of occurrences of actions *count* and *recount* executed by both instances should be the same. However, the two systems never communicate. Hence, without providing additional mechanisms to synchronize *System1* and *System2*, the description of Figure 14 cannot be implemented.

If a user considers that Message Sequence Charts present behaviors at a certain abstraction level, this kind of description is not a real problem, as using additional messages in implementations of this description is allowed. Now, if the MSC description is considered as complete, i.e. all messages actions, timers and so on that will be used by any implementation appear in the description, then some MSC descriptions cannot be implemented.

A general approach to implement a MSC description is to implement the behaviour of each instance separately (for instance with SDL) [Khendek99]. However, it has been shown that not all MSC descriptions can be implemented this way [Alur05], as some additional unspecified behaviors appear in the generated implementation. When a MSC description can be implemented by separating all instances behaviors, it will be called a *realizable* MSC.

Hence, a natural question that arises for a given MSC description M is:

- (Rez) is M realizable ?

In general, there is no procedure to answer the realizability question [Alur05]. However, recent results [Genest02, Genest04, Helouet00] have shown that a slight modification of the messages contents can allow for the implementation of some subclasses of MSCs.

2.2 General undecidable results

A problem whose answer can be yes or no is called *decidable* when there exists an algorithm that can compute a correct answer for any instance of this problem. If such algorithm does not exist, the problem is said to be *undecidable*.

- *(Mc1)* $L(M) \subseteq R\phi?$

- *(Mc2)* $R\phi \subseteq L(M)$?

- *(Mc3)* $L(M) \cap R\phi = \varnothing$?

- *(EquivL)* $L(M1) = L(M2)$?

- *(RefL)* $L(M1) \subseteq L(M2)$?

- *(IntL)* $L(M1) \cap L(M2) = \varnothing$?

- *(EquivF)* $F(M1) = F(M2)$?

- *(RefF)* $F(M1) \subseteq F(M2)$?

- *(IntF)* $F(M1) \cap F(M2) = \varnothing$?

- *(Rez)*   *is M realizable ?*

are undecidable problems. This does not mean that model checking, comparison or implementability of MSCs are always untractable problems for MSCs (i.e. they have no algorithmic solution), but rather that when considering a target application for an MSC description, users have to make sure that their specification meets some syntactical requirements. Some simple syntactic criteria allow for the characterisation of several kinds of MSC descriptions (or MSC subclasses) that enable the decidability of some of the problems listed above.

2.3 Syntactical description of MSC subclasses

Due to the undecidability result cited in section 3, several applications could be considered as impossible for Message Sequence Charts. Several restrictions to the use of MSC constructs have been defined. This section lists three of them, and for each syntactical subclass lists the possible applications.

2.3.1 Regular MSCs (RMSCs)

The set of runs of a regular MSC forms a regular language. This means that this set can be represented by a finite state machine, but also that usual techniques of model checking can be applied to regular MSC. An MSC description forms a regular MSC description if, in all loops that can appear in the description, all messages that are sent are acknowledged, either directly or indirectly, and if the body of the loop does not form disconnected parts of behaviors.
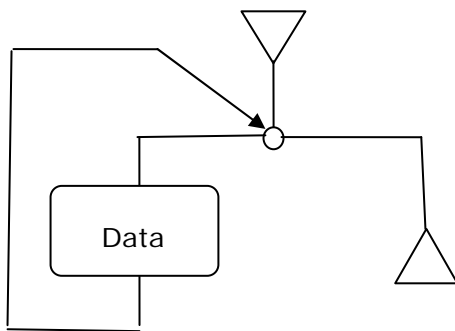
mscdocument NonRegular;

msc SimpleLoop;

expr L1;

L1: connect seq (L2 alt L3);

L2 : Data seq (L1);

L3 : end;

endmsc;
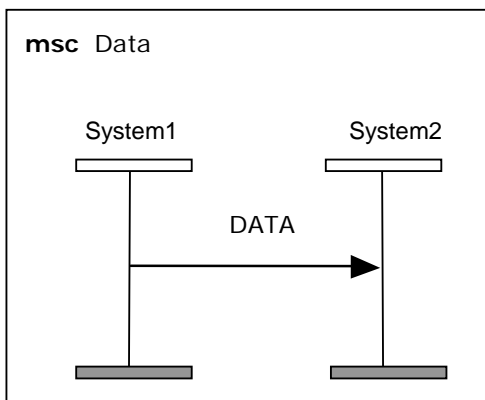
msc Data;

   instance System1;

      out DATA **to** System2;

   endinstance;

   instance System2;

      in DATA **from** System1;

   endinstance;

endmsc;

endmscdocument;

Figure 9 - A non-regular MSC

Consider for instance the MSC description of Figure 9. In Message Sequence Charts, messages are considered asynchronous. This specification then describes a protocol where instance *System1* does not have to wait for an acknowledgement of *DATA* messages before sending the next message. Runs of this MSC cannot be depicted by a finite state automaton. The second condition is illustrated by the MSC *Count* of Figure xxx. In this MSC description, all MSCs in *F(Count)* contain the same number of occurrences of atomic actions *count* and *recount*. The runs of this MSC cannot be described with a finite state automaton. The MSC Acknowledge of Figure 10 fulfils the conditions to be regular.
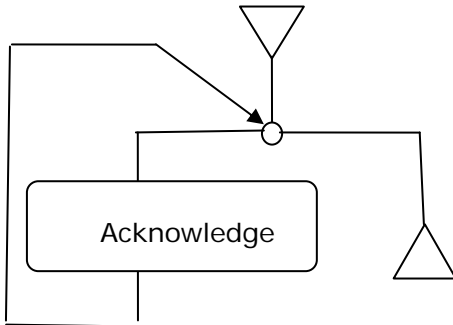
```
mscdocument Regular;


msc SimpleLoop;

expr L1;

L1: connect seq (L2 alt L3);

L2 : Acknowledge seq (L1);

L3 : end;

endmsc;


msc Acknowledge;
    instance System1;
        out Interrupt to System2;
        in Ack from System3;


    endinstance;


    instance System2;
        in Interrupt from System1;
        out Stop to System3;
    endinstance;


    instance System3;
        in Stop from System2;
        out Ack to System1;
    endinstance;
```
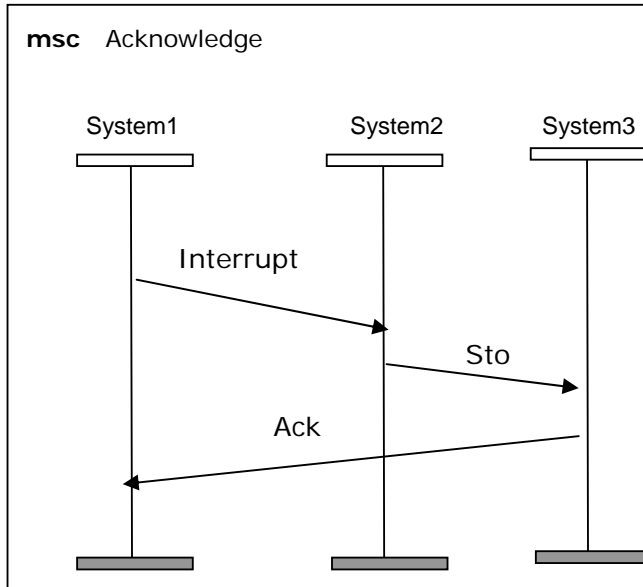

Figure 10 – A regular MSC

To summarize, the following questions might be solved by appropriate algorithms for the class of regular MSCs : *Mc1, Mc2, Mc3, EquivL,RefL, IntL.*

2.3.2 Local choice MSCs (LMSCs)

An MSC description is called a local choice MSC when, for all alternatives, there is one single instance which can take the decision of how to continue interactions with other instances.

Consider the example of Figure 11. MSC *Local* is a local choice MSC, as for both behaviors in the alternative, instance *System1* chooses how the interaction will continue, by sending a message to *System2*. MSC *Nonlocal* in Figure 12 is not a local choice MSC, as the decision to perform one of the alternative scenarios can be taken either by *System1* or by *System3*. There is a chance that an implementation of such scenario leads to a deadlock situation, if the program implementing *System1* behaves as in the first part of the alternative, and the program implementing *System3* behaves as in the second part of the alternative.
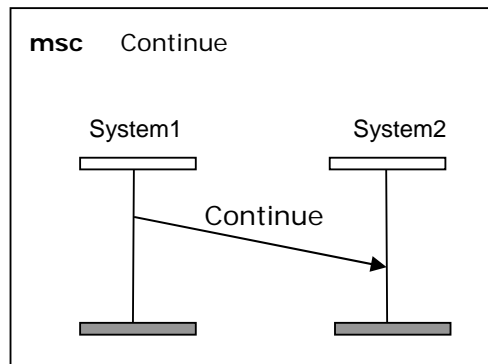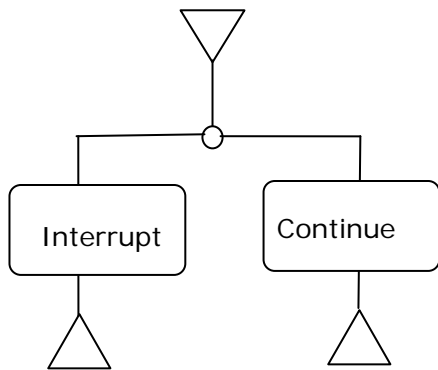
mscdocument Local;

msc Local;

expr L1;

L1 : connect seq (L2 alt L3);

L2 : Interrupt seq (L4);

L3 : Continue seq (L5);

L4 : end ;

L5 : end ;

Endmsc

Msc Interrupt ;

   instance System1;

      out Interrupt to System2;

   endinstance;

   instance System2;

      in Interrupt from System1;

      action Stop;

   endinstance;

endmsc;

Msc Continue;

   instance System1;

      out Continue to System2;

   endinstance;

   instance System2;

      in Continue from System2;

   endinstance;

endmsc;

endmscdocument;

Figure 11– a Local choice MSC.
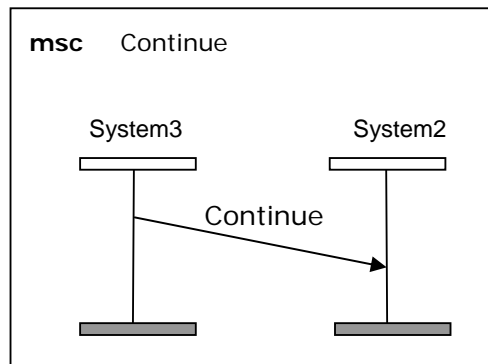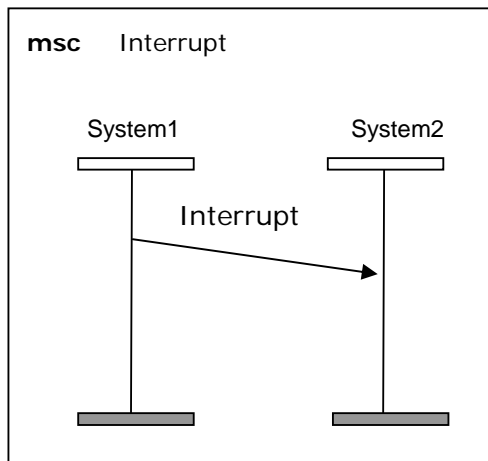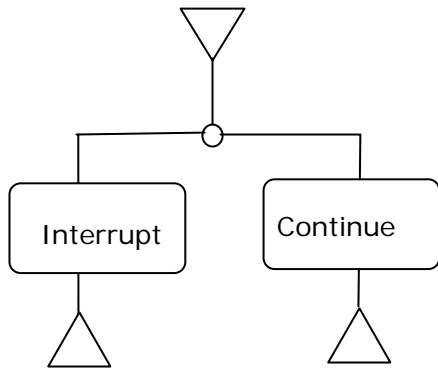
```
 Mscdocument Nonlocal;

msc nonLocalHMSC;

expr L1;

L1 : connect seq (L2 alt L3);

L2 : Interrupt seq (L4);

L3 : Continue seq (L5);

L4 : end ;

L5 : end ;

Endmsc

Msc Interrupt ;

    instance System1;

        out Interrupt to System2;

    endinstance;

    instance System2;

        in Interrupt from System1;

        action Stop;

    endinstance;

endmsc;


Msc Continue;

    instance System2;

        in Continue from System3;

    endinstance;

    instance System3;

        out Continue to System2;

    endinstance;

endmsc;

endmscdocument;
```

Figure 12 – a non Local choice MSC.

Note that local choice property is not purely local to alternatives. Consider for instance the example of Figure 13. According to the semantics of MSCs [Reniers99, Z120 AnnexB], instance *System3* can decide to send message Acknowledge without waiting for the decision of *System1*. However, if message *Acknowledge* is sent, this means that nothing occurs in the alt frame on instance *System3*, and then that the first behaviour of the alternative is ruled out.

An important property of Local choice MSCs is that they can be implemented, provided some additional control information is added to the contents of messages that are exchanged between instances. For more information, read [Helouet00, Genest02]. Local choice MSCs are a subclass of globally cooperative MSCs described hereafter.
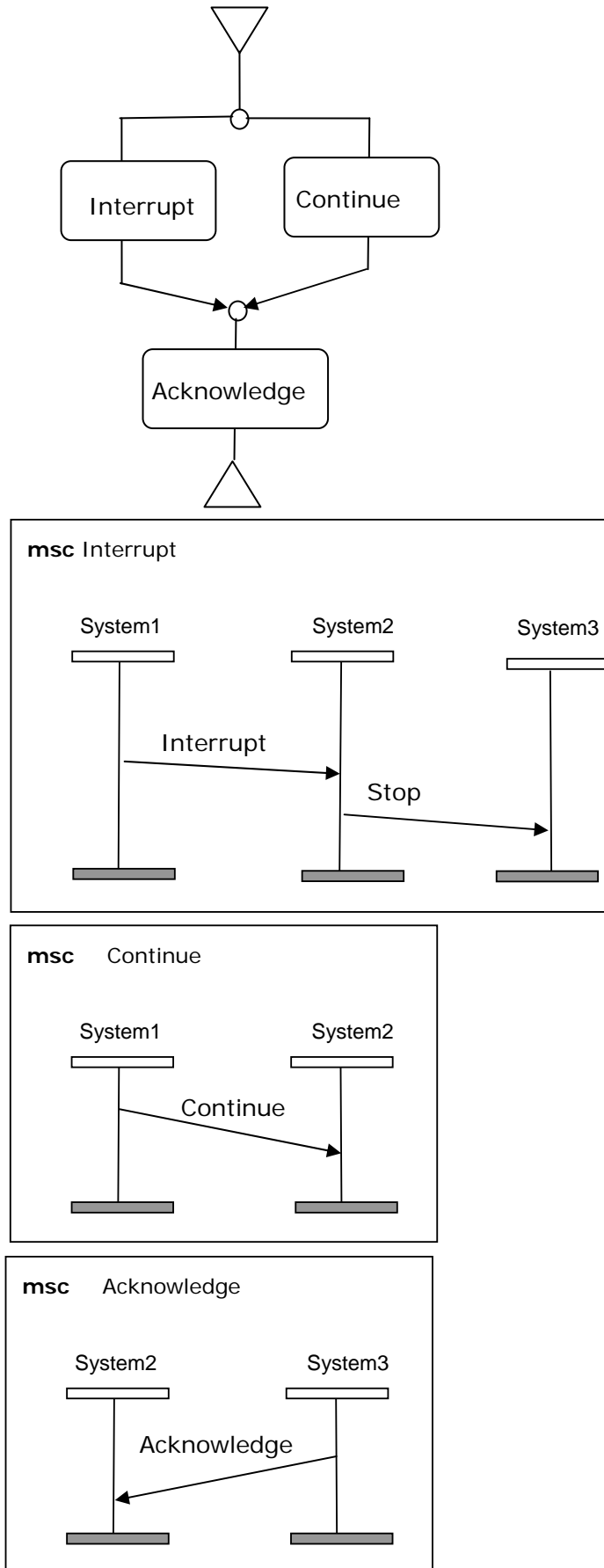
Mscdocument Nonlocal2;

msc NonlocalHMSC;

expr L1;

L1 : connect seq (L2 alt L3);

L2 : Interrupt seq (L4);

L3 : Continue seq (L4) ;

L4 : connect seq (L5) ;

L5 : Acknowledge seq (L6);

L6 : end;

Endmsc;

Msc Interrupt

   instance System1;

      out Interrupt to System2;

   endinstance;

   instance System2;

      in Interrupt from System1;

      out Stop to System3;

   endinstance;

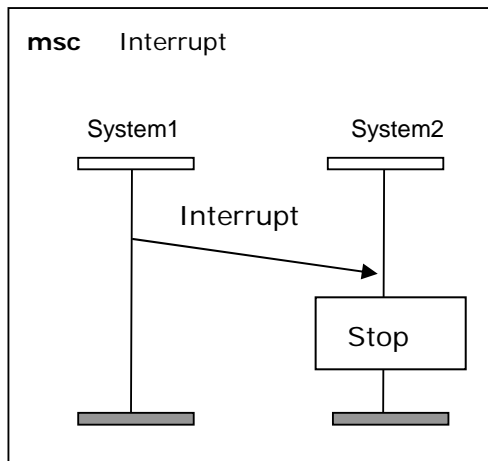   instance System3;

      in Stop from System2;

   endinstance;

endmsc;

Msc Continue;

   instance System2;

      in Continue from System3;

   endinstance;

   instance System2;

      out Continue to System2;

   endinstance;

endmsc;

Msc Acknowledge;

   instance System2;

      in Acknowlegde from System3;

   endinstance;

   instance System3;

      out Acknowlegde to System2;

   endinstance;

endmsc;

endmscdocument;

Figure 13 – a Non Local MSC.

2.3.3 Globally cooperative MSCs (GCMSCs)

An MSC description is globally cooperative if, in all loops that can appear in the description, the body of the loop does not form disconnected parts of behaviors running on distinct groups of instances. MSC *Counting* in Figure 8 is not a globally cooperative MSC: the part of the MSC enclosed in the loop frame contains two atomic actions located on different instances. MSC *Data* of Figure 4 is globally cooperative.

For two globally cooperative MSCs *M1* and *M2*, the following properties are decidable:

- *(EquivF)* $F(M1) = F(M2)$?

- *(RefF)* $F(M1) \subseteq F(M2)$?

- *(IntF)* $F(M1) \cap F(M2) = \varnothing$?

When considering two MSC descriptions *M1* and *M2*, whenever *M2* is globally cooperative the following problems have an algorithmic solution.

- *(EquivF)* $F(M1) = F(M2)$?

- *(RefF)* $F(M1) \subseteq F(M2)$?

There are also generic implementation procedures for globally cooperative MSCs [Genest04], but the drawback is that the obtained implementations can contain deadlocks, which is in general an undesirable property of a system.
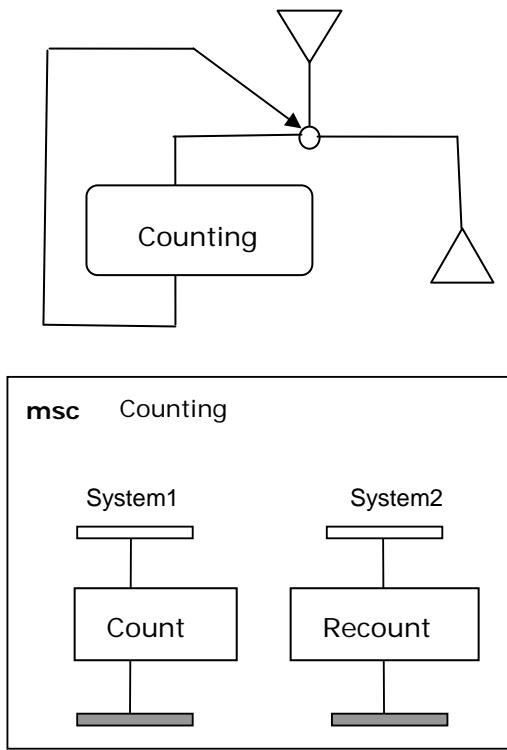
Mscdocument NotGloballyCooperative;

msc SimpleLoop;

expr L1;

L1: connect seq (L2 alt L3);

L2 : Counting seq (L1);

L3 : end;

endmsc;

msc Counting;

   instance System1;

      action count;

   endinstance;

   instance System2;

      action recount;

   endinstance;

endmsc;

endmscdocument;

<u>Figure 14 - A non Globally Cooperative MSC</u>

2.3.4 Simulable MSCs

This class is tightly related to the simulation mecghanisms used by SOFAT. A HMSC is not simulable if there is a non local choice with a branch and a loop that gets back to this choice, and an instance that appears in the branch and not in the loop. When a HMSC is simualble, the SOFAT can compute a graph grammar, and a normal form for this grammar, and use it to perform a simulation of the HMSC.
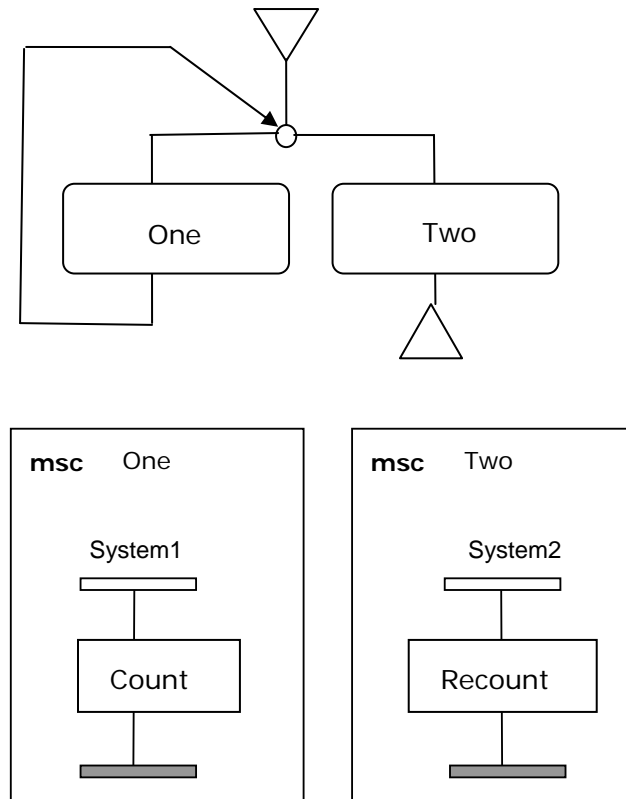
Figure 15 A non- simulable HMSC

2.4 Bounds

As described in part 1 of this document, MSCs have an interleaved semantics, that is they define sets of runs, or sequences of events. The intuition behind Message Sequence Charts is that a scenario description defines the behaviors of a distributed system, and more precisely of entities that communicate via channels. These channels can be seen as buffers that store message contents until reception by the destination. Two question then naturally arise on the size of these buffers :

Is there a bound k such that in every execution of my MSC, all buffers necessarily contain less than k+1 messages ? This property is called universal boundedness.

Is there a bound k such that for every basic MSC generated by my MSC description, there is one execution of this MSC in which the contents of all buffers never exceeds k messages? This property is called existential boundedness.

2.5 Simulation

Simulation of a HMSC in SOFAT V3 relies on a representation of HMSC runs by a graph grammar, which avoids complex interleaved model. When a HMSC is simulable, then this grammar has a normal form, that allows for the comparison of HMSC descriptions, and for simulation. For more details on the simulation framework based on graph grammars, we refer the reader to [Helouet99].

PART 3 : SOFAT

INSTALLATION

PROJECT MANAGEMENT with SOFAT

Main functionalities

3 SOFAT

3.1 Installation

SOFAT is distributed as a self-installing archive. Double click on the archive file to start the installing process. This opens a window on the screen. Press next button to continue the installation, or on the "CANCEL" button to stop the installation. The next state of the installation window shows you the license conditions for SOFAT. SOFAT is distributed freely, on an AS IS basis, without any warranty. If you agree with the terms of the licence, then press the "ACCEPT" button to continue the installation.

Once you have accepted the terms of the licence, you can select the installation directory. By default, this directory is : C:\Program Files\SOFAT. You can change the directory by typing an new directory name, or by browsing the existing directories on your disk.
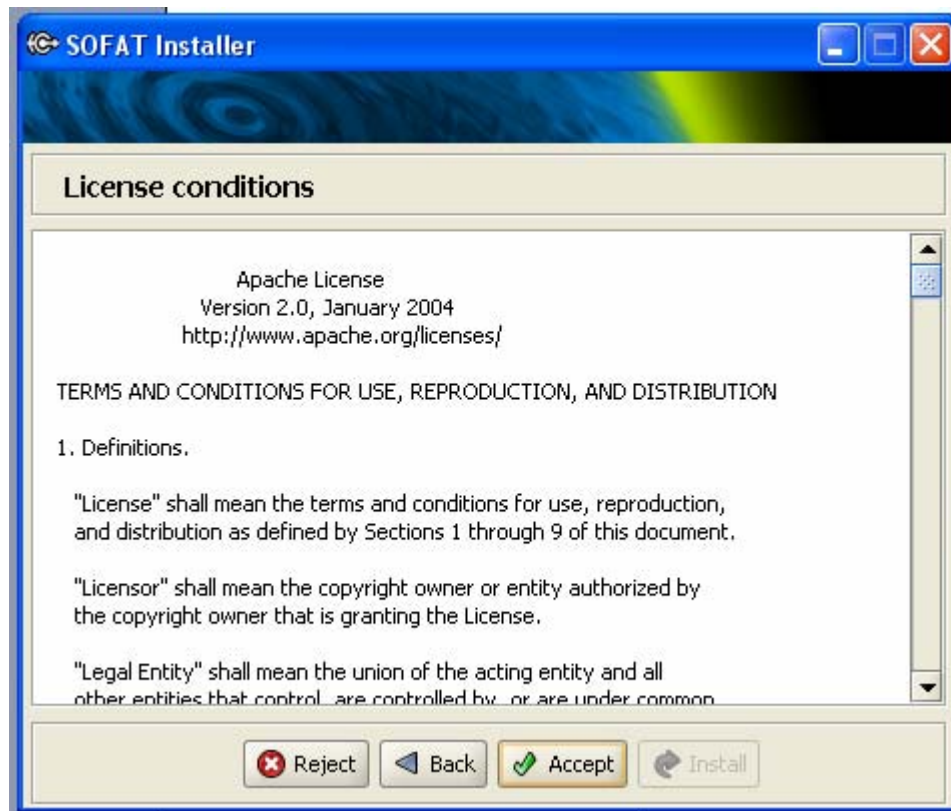
Figure 16 – The installer window

Chose button Next to continue the installation. If the chosen directory does not exist, then you will be asked whether you want to create it. In case of positive answer, the new directorywill be created by the installer. You do not have to chose the components to install, just press the "Next" button. You can then show the details of the installation such as the output of the installer and the error messages. Then press Install button to continue. The installation consists in the creation of a lib directory in the main folder, and in the creation of an executable jar file located in the lib directory. If you have installed SOFAT V3 in directory C:\Progam Files\SOFATV3, then this archive should be located at C:\Progam Files\SOFATV3\lib\sofat.jar.

SOFAT is now ready, and you can run it with a double click on the SOFAT.jar executable archive. The first execution of SOFAT will create two new subdirectories, .nullSofatDebug and SampleProject, that are

used for debugging purposes, and a preference file Preferences.txt, that contains the saves user preferences.

For more details on the architecture of a SOFAT project and on the temporary or input files created by the software, please consult section XXXXX.

3.2 Running SOFAT V3

SOFAT is distributed as a jar file named SOFAT.jar, located in the lib directory situated in the installation directory. If you have installed SOFAT V3 in directory C:\Progam Files\SOFATV3, then this archive should be located at C:\Progam Files\SOFATV3\lib\sofat.jar.

If you have installed a java Virtual Machine on your computer, a double click on the archive should launch SOFAT. SOFAT has been tested on WINDOWS machines, but as long as a recent enough Java virtual machine is installed on your computer, the archive should be usable.

3.3 Project Management

A project in SOFAT is directory for storing .hmsc files which are used to specify scenarios. The analysis results are stored in the Result directory within the particular project directory. The temporary files that are created by SOFAT are stored in a subdirectory Temp of the project directory. These project ARE NOT symbolic views of the physical files and directories. Appending and HMSC file to a project then means creating a new copy of the file in the corresponding directory, and deleting a project means deleting the corresponding directory and all its contents.

3.4 Main Functionalities

3.4.1 MSC projects edition

SOFAT can be used to create MSC projects, that contain hmsc files, results of analyses, reports, graphical presentation of MSCs, etc. These files are gathered into a project. A project is mainly a directory containing the hmsc descriptions and all files computed by SOFAT. These project ARE NOT symbolic views of the physical files and directories. Appending and HMSC file to a project then means creating a new copy of the file in the corresponding directory, and deleting a project means deleting the corresponding directory and all its contents.

When a project has been created, all hmcs files that it contains can be edited within a special window, saved, saved with a new name, etc. The simple msc editor provided also allows for the usual cut, copy, paste functionalities that exist in most editors.

3.4.2 Properties of MSC

When an hmsc description is open in SOFAT, the tool allows for a syntactical analysis, that detects syntax errors in the software and gives the line where this error is located with some information on the expected correct syntax. Syntax checking also analyses that all references to a basic MSC or to a HMSC point to a MSC that is also depicted in the analyzed file. Hence if a description contains a line of the form: L1: Missing seq (L2) ; and that the description contains no basic MSC or HMSC called Missing, the syntactic analyzer will display a warning. The last analyze performed during syntactic analysis is cyclic referencing in HMSCs (see Part 1 of the document for details).

When the syntactic analysis of a HMSC description is done, the analysis of its formal properties can be performed. The analyzer detects if a HMSC description is regular, globally cooperative, simulable, and computes the existential bounds for each basic MSC in the description. For more detail on these formal properties, please consult PART 2 of this document.

3.4.3 Atoms and MSC splitting

As mentioned in section xxx long behaviors can alternatively be described as a large basic MSC, or as sequences of smaller. SOFAT V3 allows the partition of a BMSC into an equivalent sequence of atomic bMSCs.

3.4.4 Exploration/simulation

When a HMSC description is simulable, SOFAT allows for functionalities such as interactive simulation, exploration (exhaustive up to a certain limit of the state space).

3.5 SOFAT Panel/Windows

The SOFAT interface consists of the following five windows each with a specific functionality. Figure 17 shows a screen captire of SOFAT V3, with its 5 different panels. On the picture these panels are organized as depicted in the schema below

Menus		
1 Project panel	2 Msc panel	3 Editor panel
4 Result panel		5 Log panel

The following paragraphs list the functions of each panel.
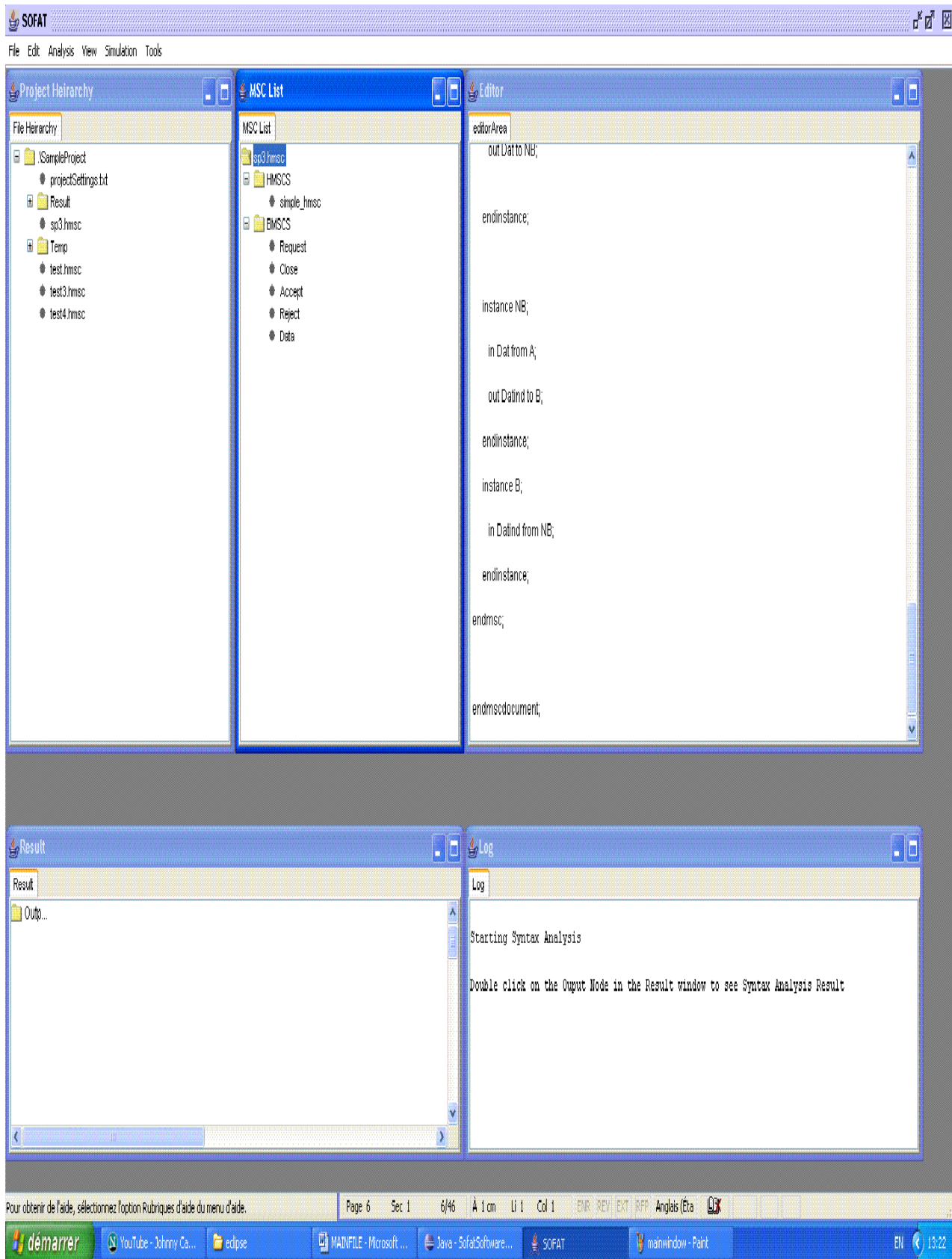
Figure 17 - SOFAT window showing different Panels

3.5.1 Project Hierarchy panel

The role of the Project Hierarchy panel is to display the entire project directory, i.e. the files within the project. It is possible to perform some actions on the items that are listed in this window. A file can be opened in by clicking on it. By right clicking on the individual files or directories the following functions can be performed:

Cut : The selected file is cut and can be paste into another project/directory. The file is deleted from the current directory. The project hierarchy is updated after the "Cut" action is performed.

Copy: A file can be copied and paste into a directory. The directory to paste into can be selected by clicking on the "Paste" option.

Paste: When the paste option is clicked, the user is required to select the directory where to paste the file.

Add Msc file: this option can be executed only when a project directory node is selected. It gives the user the possibility to import an MSC from another directory into the existing project. The imported file is copied into the physical directory of the project.

Delete: the selected file is removed from the project. The file projectSettings.txt should not be deleted.

Open Outside Sofat - The file can be opened outside the SOFAT software. This can be used to work on multiple options simultaneously.

3.5.2 MSC List-

When a existing hmsc file is opened or a new file is created, the MSCs list is displayed in this window. The MSCs are divided into HMSCS and BMSCS. The following functions can be performed on the MSCS.

Split MSc : This option can be used only on a BMSC. It splits the BMSC to its constituent atoms. The user can save just the BMSC in a separate file or the entire hmsc document with the split msc in a separate file or overwrite the existing file with the split basic msc.

Graphical View of MSC : The user can use this option to generate a graphical view for a BMSC or for a HMSC. The most recently generated file can also be viewed. It also possible to view previously generated file by clicking on the other option.

Separate file for msc : A BMSC or HMSC can be exported into a separate hmsc document. This functionality allows for the exportation of a part of a description (which otherwise would necessitate to copy the wanted part, create a new document, paste, and save under the desired name). When a BMSC is converted to a separate document, a new hmsc document is created. It contains a simple HMSC that refers to the exported bMSC. This way, exported bMSC produce syntactically correct msc documents.

File Edit Analysis View Simulation Tools

Project Heirarchy

File Heirarchy

- C:\Deepak\Examples
  - abc.hmsc
  - accept.hmsc
  - acp.hmsc
  - acpp.hmsc
  - ap.hmsc
  - apc.hmsc
  - bounded.hmsc
  - cfc.hmsc
  - cfc2.hmsc
  - choseOp.hmsc
  - close.hmsc
  - co.hmsc
  - confluent.hmsc
  - cycles.hmsc
  - cyclicref.hmsc
  - dat.hmsc
  - datat.hmsc
  - double_conn.hmsc
  - Emprunt2.hmsc
  - emprunt3.hmsc
  - emptycycle.hmsc
  - Examples
  - exemple_tewfik.hmsc
  - hier.hmsc
  - incorrect_match.hmsc
  - incorrect_match.hmsc~
  - instance_problems.hmsc
  - jk_attention.hmsc
  - m1.hmsc
  - m2.hmsc
  - m4.hmsc
  - multicycles.hmsc
  - newhmsc_1
  - nonlocal.hmsc
  - nonlocal2.hmsc
  - nonstarcon.hmsc

MSC List

MSC List

- newhmsc_1
  - HMSCS
    - NewTest
  - BMSCS
    - DisplayMap
    - ChoseOp
    - LoadMap

Editor

editorArea

```
I3: ChoseOp seq (I1);
I4: LoadMap seq (I5);
I5: end;
endmsc;

msc DisplayMap;
/* jjhfshfks
gjhgjgjjhhg
*/
 inst User, Server;
 instance User;
out map to Server;
 endinstance;
 instance Server;
in map from User;
 endinstance;
endmsc;

msc ChoseOp;
inst User, Server;  /* fjhghjgjhg gjhgjhg hgj*/
instance User;
out operation to Server;
 endinstance;
 instance Server;
in operation from User;
 endinstance;
```

Split

○ Only Split MSC as a Separate File

○ Entire hmsc document(including split msc) as a separate File

○ Overwrite Existing File

| :\Deepak\Examples\dsmap.hmsc | Browse | OK |

```
endmscdocument;
```

Result

Result

- Output
  - Syntax Analysis
    - Correct Syntax
    - References Match
    - All bMSC checked
  - Property Analysis
    - Not Regular
      - Communication Graph is not Strongly Connected Components
    - Globally Cooperative
    - Non Local Choice Node: {I1} in msc NewTest
    - Can't Simulate
    - Bounds

Log

Log

```
Starting Syntax Analysis

Double click on the Ouput Node in the Result window to see Syntax Analysis Result
Weak sequence built

Double click on Ouput Node in the Result window to see Property Analysis Result
```

Pour obtenir de l'aide, sélectionnez l'option Rubriques d'aide du menu d'aide.
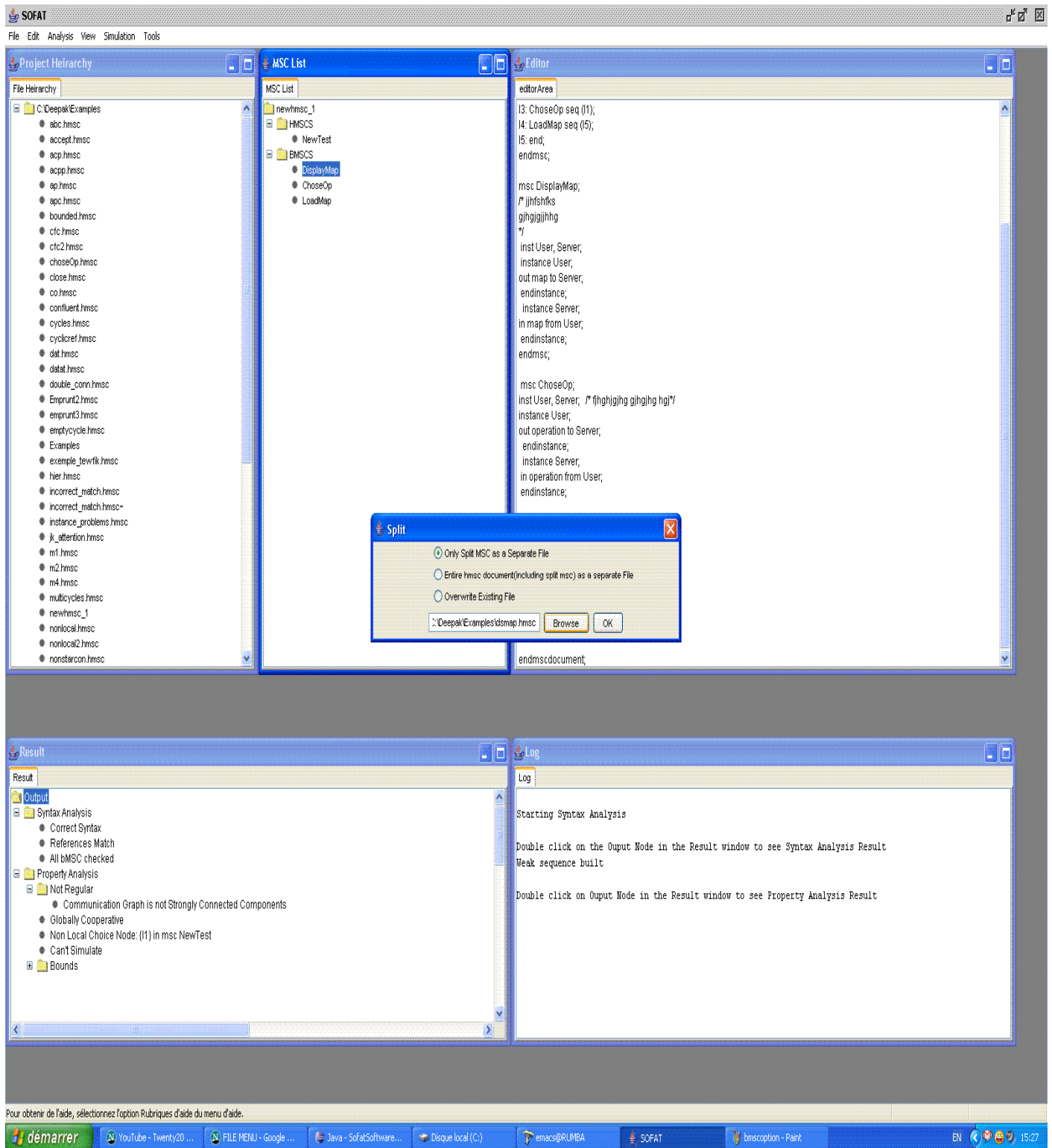
Figure 18: Spilt MSC option

3.5.3 Editor Area

The role of the editor area is to provide a micro-text editor with functionalities dedicated to HMSC. When an hmsc document is opened, it is displayed in this window. It is possible to perform standard editing operations such as cut, copy, paste and find on the contents of the editor area. In addition to this, the tool menu allows for highlighting of MSC reserved keywords.

3.5.4 Result

The Result window displays the result of the analysis i.e. Syntax analysis and Property analysis in a hierarchical form. The results are displayed in a hierarchical fashion.

For the syntax analysis, the reported errors are syntax errors, MSCs that are references but not used, messages that are sent but not received in basic MSCs, etc. When a syntax error occurs, a error message indicating the line of the error and the expected keyword in also displayed.

For the property analysis, the window lists all formal properties of the considered hmsc (regular, simulable, globally cooperative, local choice), and the existential bounds on each basic MSC.

3.5.5 Log

The log window is used to display messages when various features of the software are used. It indicates whether the function was performed correctly or not and where the user can see the output. If you consider the build grammar option, once the grammar is built, the log window displays the name of the grammar file and the location where it is saved. The log area messages can be cleared.

3.6 SOFAT Menus

3.6.1 FILE MENU

The file menu consist of the following menu items. Below each menu item is explained in detail.

New

Project

MSC

The new menu item allows for the creation of a new project and a new msc within an existing project. When the new project is created the file projectSettings.txt is created within the directory. The file has details such as project name, date of creation of the project, type of project and the supported files. The user must not delete this file as when opening a project, the system checks for this file. A new MSC file contaning an hmsc document can be created for an existing project. When creating a new project or a new MSC file, the system checks to see if the current file is already saved or not and acts accordingly.

Open

Project

MSC

It is possible to open an existing project. As mentioned previously while opening a project the system checks for the existence of the file projectSettings.txt. An existing MSC within the project can be opened as well. In case the user opens a file outside the current project, the current project is closed and the project of which the file was part is opened. When opening a project or a MSC file, the system checks to see if the current file is already saved or not and acts accordingly.

Save

The file is saved with the existing name.

SaveAs

An existing file or a new file can be saved with a new name.

ClearLog

This functionality clears the log window.

Exit

This functionality is used to exit the entire software. Before exiting, the software checks for the current file and saves it. The details such as current file, current project etc. are saved in a file called Preferences.txt. This file is read every time the project is opened. This allows the user to start working with the same file and project before exiting.

3.6.2 EDIT MENU

All of the edit functionalities are linked to the editor area. The edit menu list the following functionalities.

Cut

The following allows to cut the selected text from the editor area.

Copy

The following allows to copy the selected text in the editor area.

Paste

Any text that has been copied or cut from the editor area can be pasted in to the editor area.

Find

The find functionality allows to search for text within the editor area. The text can be searched either as case sensitive or not. Figure 19 shows that when text in a case different from that in the editor area is searched for with the case sensitive option set, the software returns "text not found" message. In case of figure 20, the word "INST" is searched with the ignore case option set. The software is able to search the text and highlight it as can be seen from the figure. In the figure, the text "INST" is searched and not found. Consider figure 21, the word "inst" is searched with the case sensitive option set, the software shows the first instance of the word by highlighting it. Once again the "Find" button is clicked it searches for the text from the previous position.
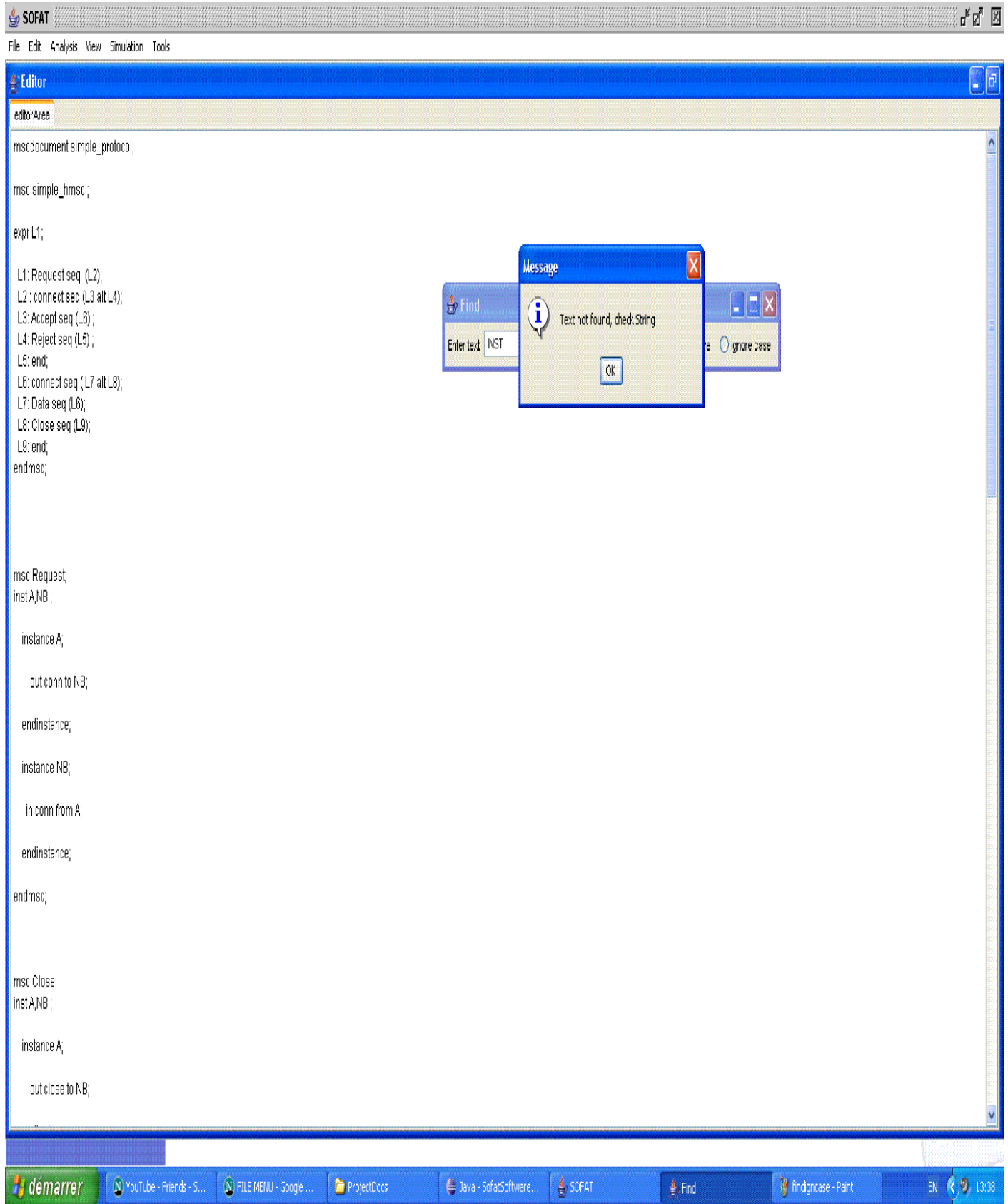
File Edit Analysis View Simulation Tools

Editor

editorArea

```
mscdocument simple_protocol;

msc simple_hmsc ;

expr L1;

 L1: Request seq  (L2);
 L2 : connect seq (L3 alt L4);
 L3: Accept seq (L6) ;
 L4: Reject seq (L5) ;
 L5: end;
 L6: connect seq ( L7 alt L8);
 L7: Data seq (L6);
 L8: Close seq (L9);
 L9: end;
endmsc;



msc Request;
inst A,NB ;

  instance A;

    out conn to NB;

  endinstance;

  instance NB;

   in conn from A;

  endinstance;

endmsc;



msc Close;
inst A,NB ;

  instance A;

    out close to NB;
```

Find

Enter text INST

Message

Text not found, check String

OK

Ignore case

Figure 19: Find option with case sensitive

SOFAT

File Edit Analysis View Simulation Tools

Editor

editorArea

```
endmsc;


msc Close;
inst A,NB ;

  instance A;

    out close to NB;

  endinstance;

  instance NB;

   in close from A;

  endinstance;

endmsc;


msc Accept;
inst A,NB ;

  instance A;

    in conf from NB;

  endinstance;

  instance NB;

   out conf to A;

  endinstance;

endmsc;


msc Reject;
inst A,NB ;

  instance A;
```

Find

Enter text INST   Find   ○ Case Sensitive ◉ Ignore case

Pour obtenir de l'aide, sélectionnez l'option Rubriques d'aide du menu d'aide.

démarrer   YouTube - Friends - S...   FILE MENU - Google ...   eclipse   Java - SofatSoftware...   SOFAT   Find   Sans titre - Paint   EN   13:23

Figure 20: Find option with case ignore option

File Edit Analysis View Simulation Tools

Editor

editorArea

```
mscdocument simple_protocol;

msc simple_hmsc ;

expr L1;

 L1: Request seq  (L2);
 L2 : connect seq (L3 alt L4);
 L3: Accept seq (L6) ;
 L4: Reject seq (L5) ;
 L5: end;
 L6: connect seq ( L7 alt L8);
 L7: Data seq (L6);
 L8: Close seq (L9);
 L9: end;
endmsc;




msc Request;
inst A,NB ;

  instance A;

    out conn to NB;

  endinstance;

  instance NB;

   in conn from A;

  endinstance;

endmsc;



msc Close;
inst A,NB ;

  instance A;

    out close to NB;
```

Find

Enter text INST   Find   ○ Case Sensitive   ⦿ Ignore case

Transferring data from sjc-v105.sjc.youtube.com...   Open Notebook

Smart Insert   120 : 52

démarrer   YouTube - Friends - S...   FILE MENU - Google ...   ProjectDocs   Java - SofatSoftware...   findcasesensitive - Paint   SOFAT   Find   EN   13:49

Figure 21:Find option with Ignore case set highlighting the text.

3.6.3 ANALYSIS MENU

The options in the analysis menu are used to perform the basic operation such as checking of correctness and view properties of these mscs. The following options are available to the user:

3.6.3.1 Syntax Analysis

The results of the syntax analysis are displayed in the Result window. They are displayed in a hierarchical fashion. In case of mistake, the reason is also mentioned. If the syntax analysis is not successful, property analysis cannot be performed. The log area displays messages whether any Syntax Error has occurred.

The reported errors are syntax errors, MSCs that are references but not used, messages that are sent but not received in basic MSCs, etc. When a syntax error occurs, a error message indicating the line of the error and the expected keyword in also displayed.

Figure 22 shows the result of a Syntax Analysis on the file sp3.hmsc which is provided in the examples folder. Figure 23 shows the result when syntax analysis is performed on the incorrect_match.hmsc file in the examples folder. It shows the message that mscs req1 and requ although have been referenced are not defined.
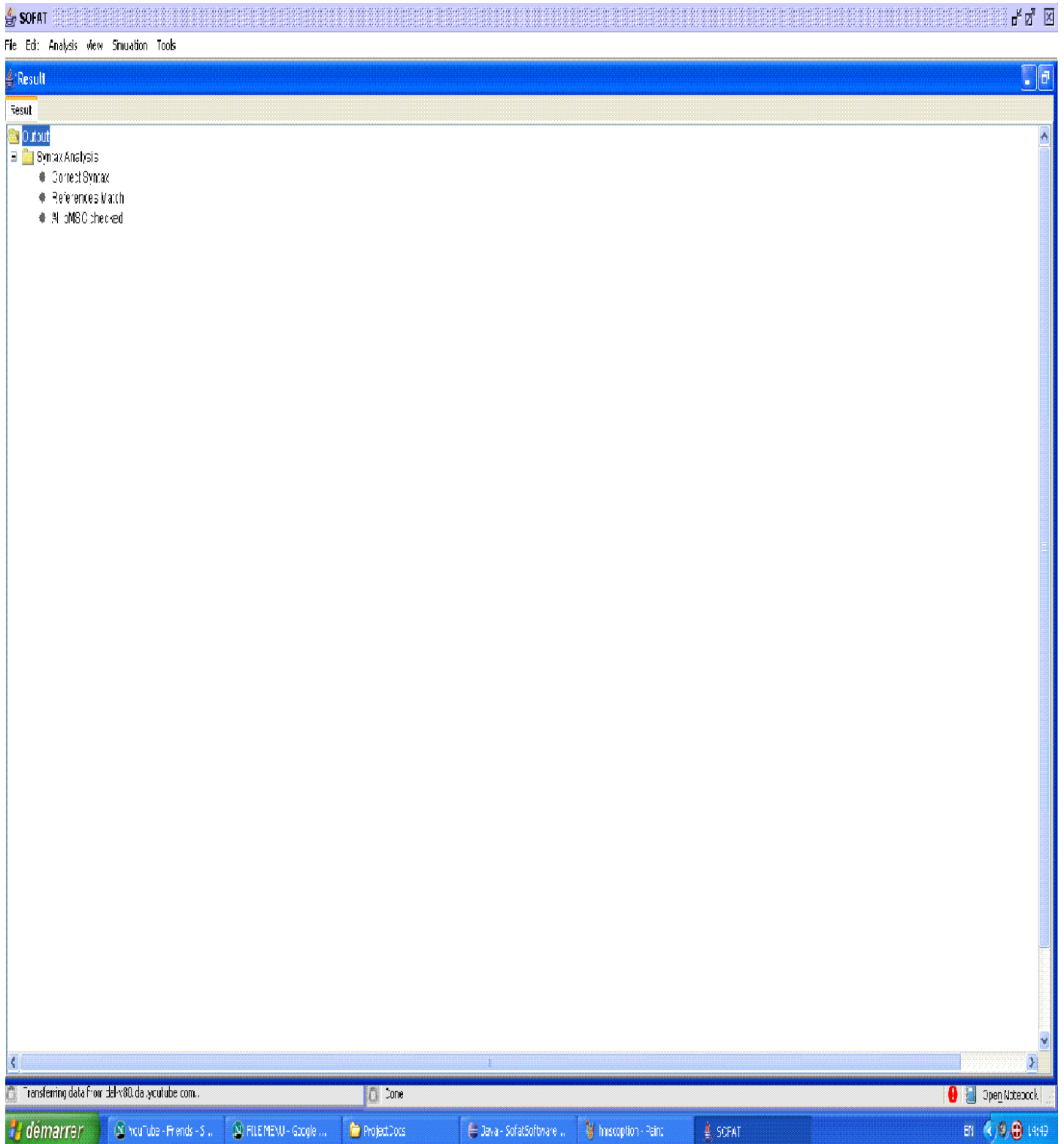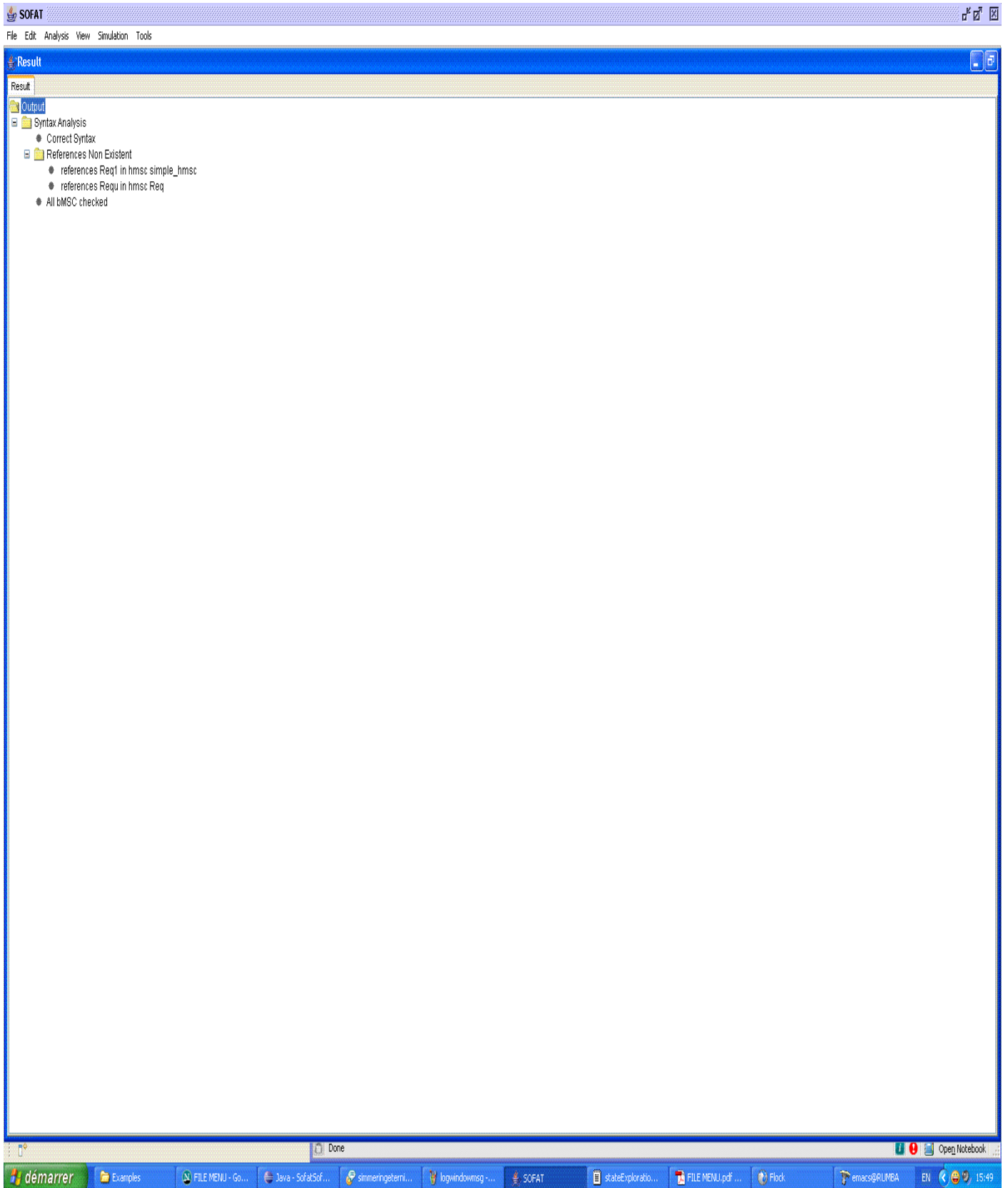
Figure 22: Syntax Analysis Result Option

Figure 23: Syntax Error

3.6.3.2 Property Analysis

Similarly to Syntax Analysis the results for property analysis are displayed in the Result window in a hierarchical form. The software determines which properties hold for the considered MSC, and which do not. When a MSC does not satisfy a given property, the reason why this property does not hold is also displayed. The property analysis analyses whether a MSC is regular, local choice, globally cooperative, and simulable, and computes the bounds for basic MSCs. For more information of properties of Message Sequence Charts, consult part 2 of this document.

Figure 24 shows the result of "Property Analysis" performed on the file sp3.hmsc This hmsc is not regular, it is globally cooperative, local choice, and can be simulated. The existential bound for all bMSCs appearing in the description is 1.
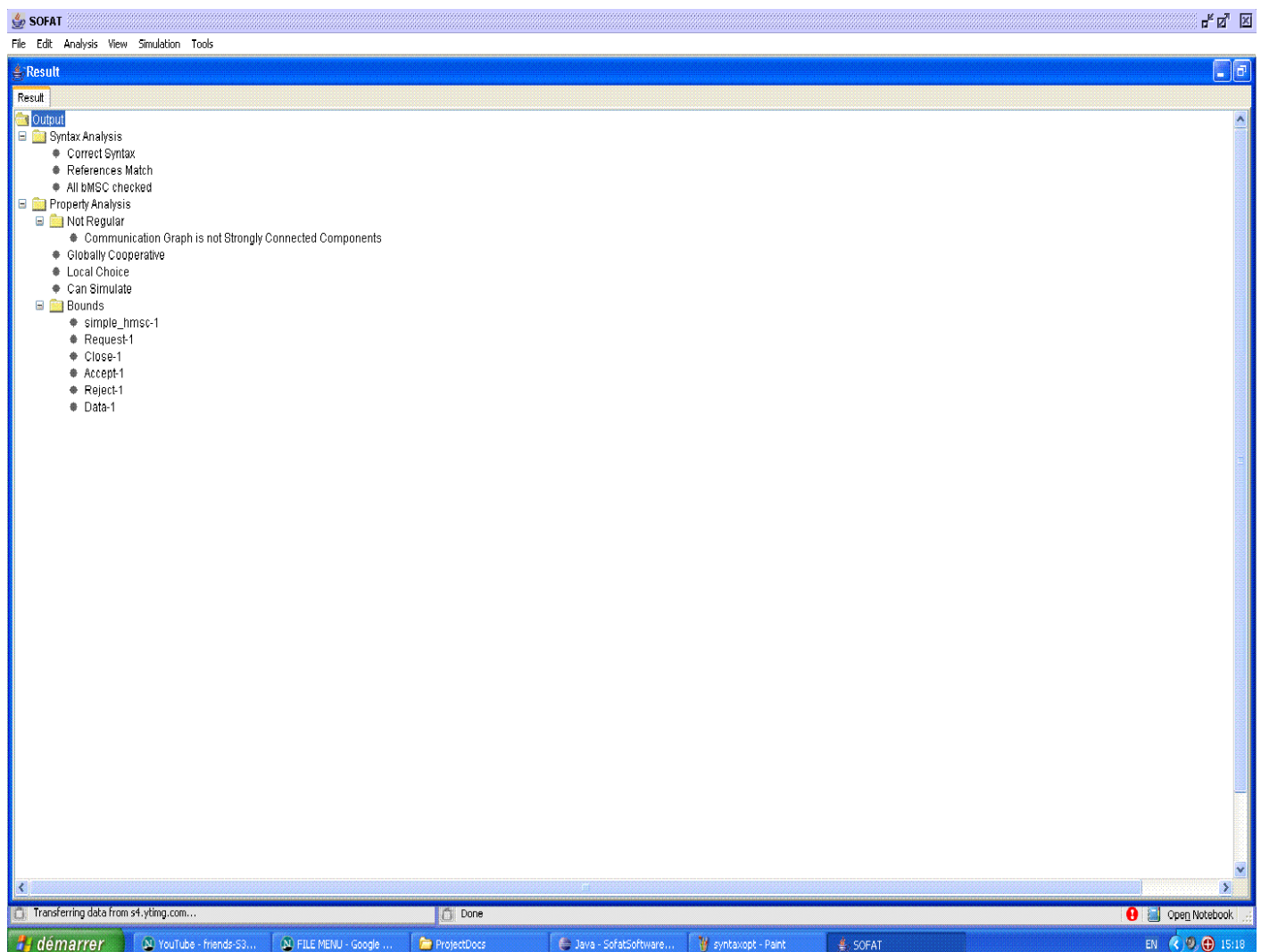
Figure 24: Property Analysis result showing in the Result window.

3.6.4 VIEW MENU

The view menu allows the user to display either a BMSC or a HMSC in a graphical form. Once the syntax analysis is performed , the items of the view menu are enabled. When a BMSC or HMSC is displayed, it is first transformed into a graph in the DOT format, and this file is then transformed by DOT in a GIF or PostScript file, depending on the user choice. The produced image is displayed in a popup window. By default, the chosen format is PostScript. The result file is saved in the Result folder.

3.6.4.1 Basic MSC

Once the option is clicked a drop down list containing the names of the basic mscs within the hmsc document is displayed. The user has to select the Basic MSC to view and click OK. Figure 25 shows how a basic MSC called Request is displayed. The user selects the basic msc to view from the list presented. In this case the user selected "Request". The Log window displays the name of the produced file and the directory where it is saved.
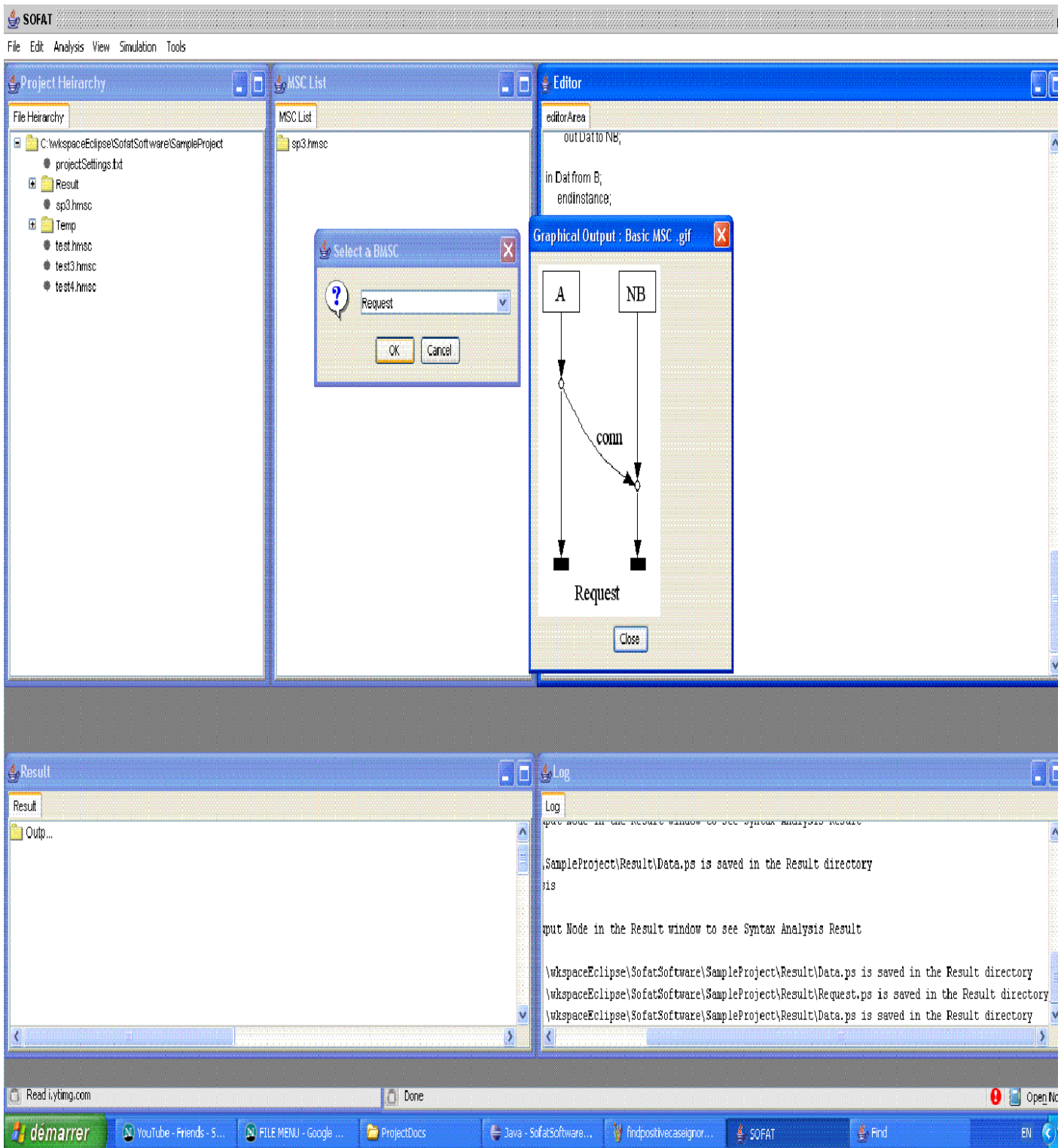
Figure 25: Basic MSC option

3.6.4.2 High Level MSC

Similar to the basic msc functionality described above, the hmscs in the document is displayed in a drop down list and the user to chose the hmsc to view.Figure 26 shows that the user selects the "simple_hmsc" HMSC for viewing. Similar to the case above the Log window displays the name of the resultant file and the directory where it is saved.
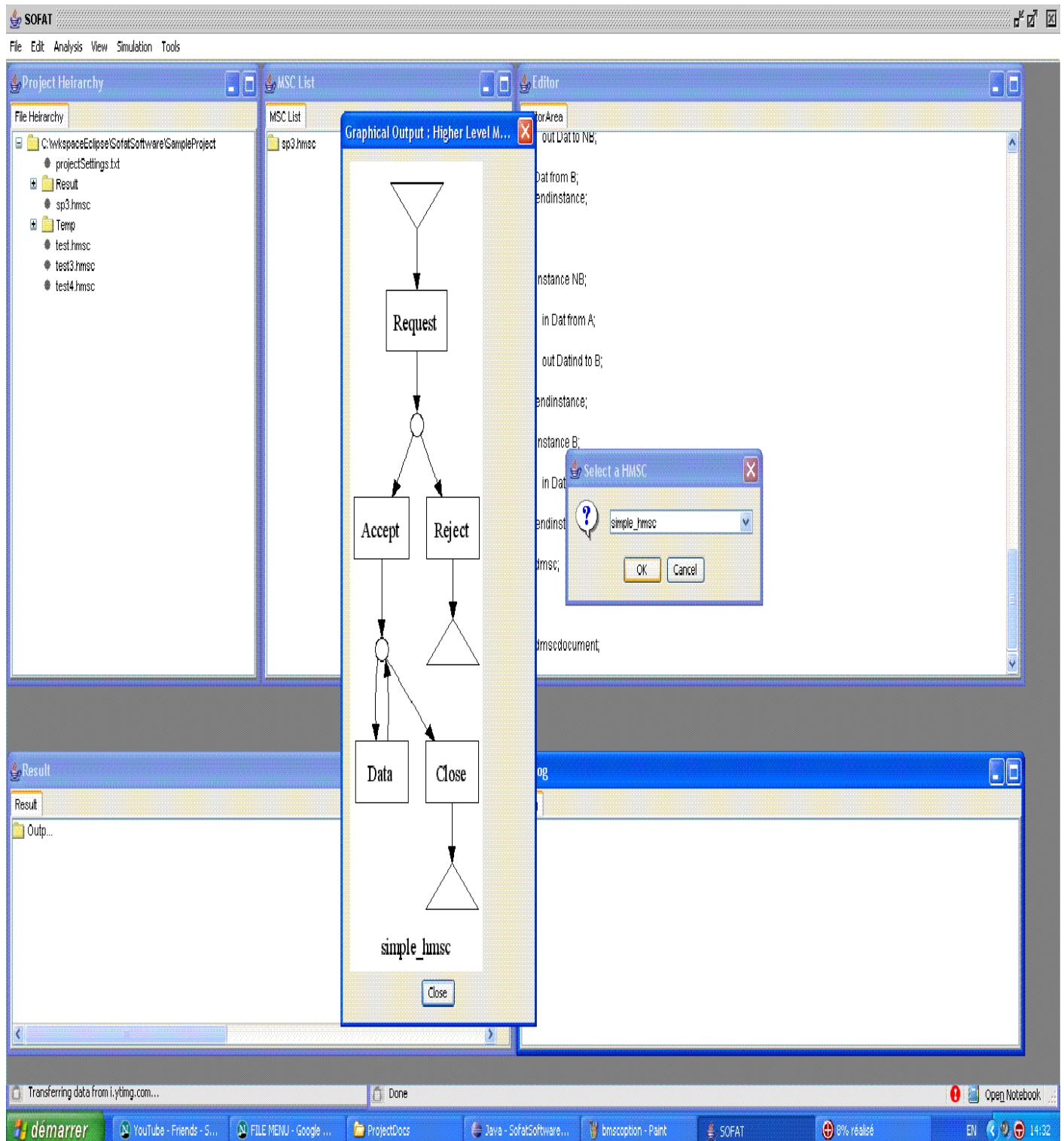

Figure 26: Higher Level MSC option

3.6.4.3 All

This item allows to save a graphical description of all basic MSCs and HMSCs within current hmsc document. In this case all the basic mscs and high level mscs are saved in a single postscript file that can be read afterwards.

3.6.5 SIMULATION MENU

Some Message Sequence Charts specifications can be simulated. The simulation framework used by SOFAT is based on the graph grammar framework of [Helouet99]. HMSCs are first transformed into a gram grammar (this translation is always feasible). The produced graph grammar is a compact representation of an event structure which configurations are exactly the set of MSCs generated by the transformed HMSC. When the considered HMSCs are simulable, the graph grammar can be normalized. This normalized grammar is then used by the simulator.

The functionalities provided here can be performed after Syntax Analysis.

Build Grammar.

The grammar file is built and saved in the Result directory. It is saved with the extension ".gram". The log area displays the name and location of the grammar file.

Normalize.

This option can be exercised only if the grammar is built correctly. The normalized grammar file has a name similar to the grammar file with a "_norm" before the extension.

Simulate.

When a grammar has been normalized, it can be used for simulation. Two windows are displayed one window is the "State" window displaying the events that can be fired in the future. The other window displays the list of events that can be fired from current state of the simulation. By clicking on an event the transition is fired, and a new state is computed, and displayed in the "State" window. Figure 27 gives a snapshot of a running simulation. For more details of the simulation framework used by SOFAT, we refer interested readers to [Helouet99].
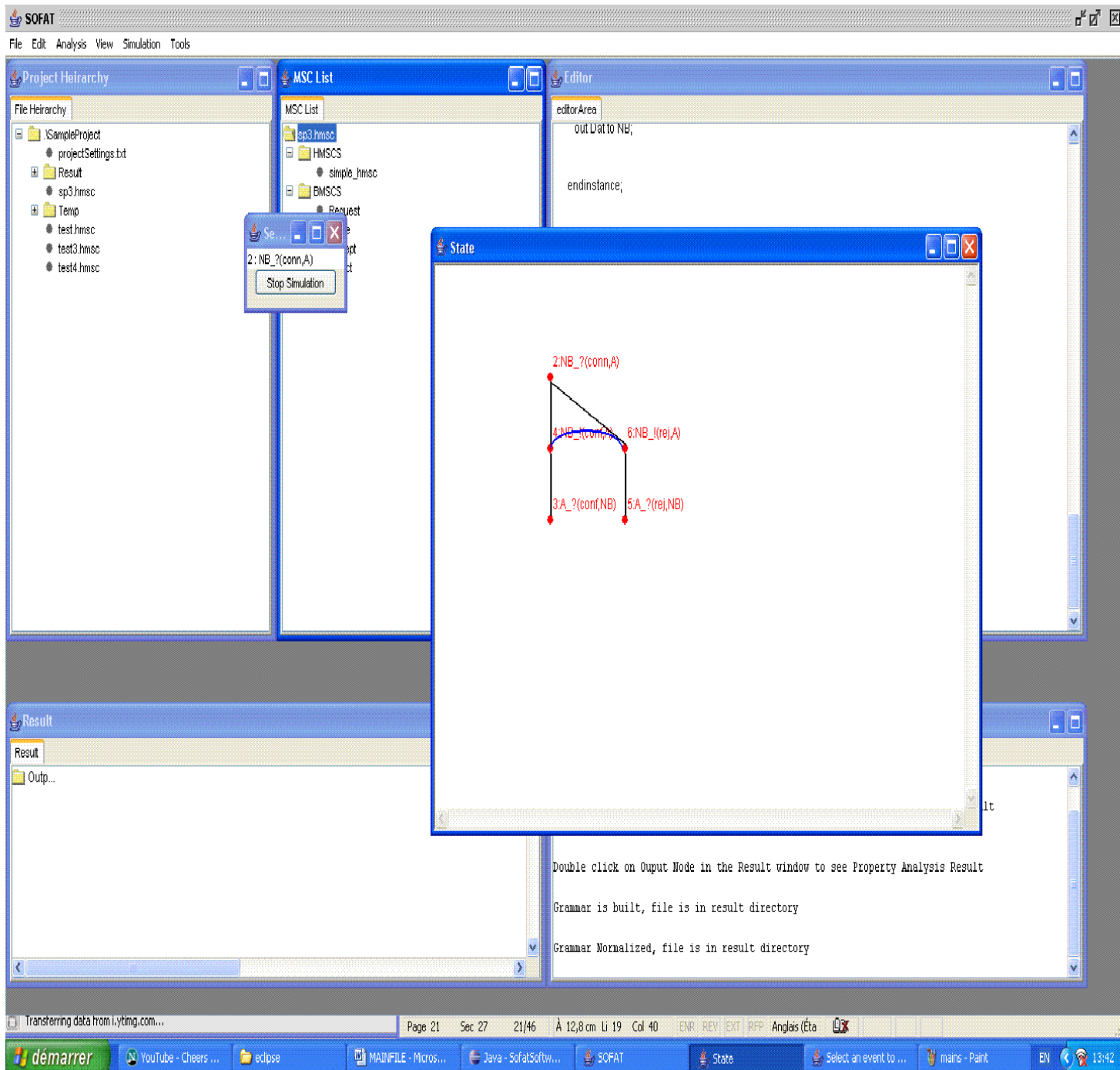
Figure 27: Simulation Window

<u>Exploration</u>.

The simulation framework can be used interactively, or automatically to produce a finite state machine. Note that the interleaved semantics of HMSCs is not necessarily a regular language, and can not always be represented by a finite state machine. Hence, for exploration, a limit to the number of states is set before exploration.

There are two options to the user to view the output either in the form of an automaton (saved as a ".aut") or a dot file (saved as a ".dot"). The file has the name of the form "stateExploration_"+ name of msc followed by the necessary extension. Figure 28 shows how an exploration is launched for a description contained in file sp3.hmsc. The user selects the options to save the states reached during exploration as an automaton file (.aut format). The maximal number of states selected by the user is 4, which means that the axploration will stop automatically when 4 states have been generated. The Log window displays the name of the resulting file and the location it is saved (Figure 29).
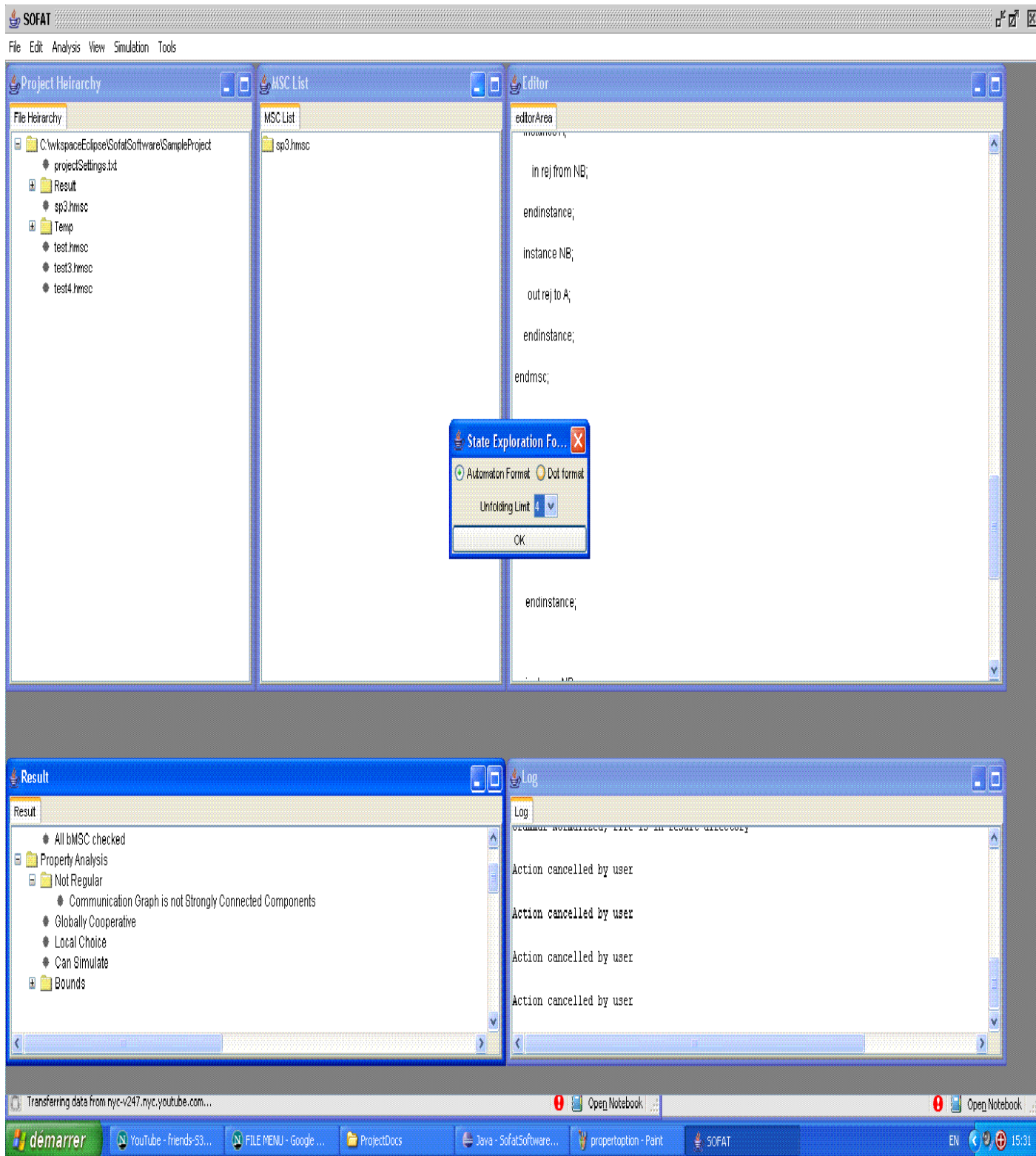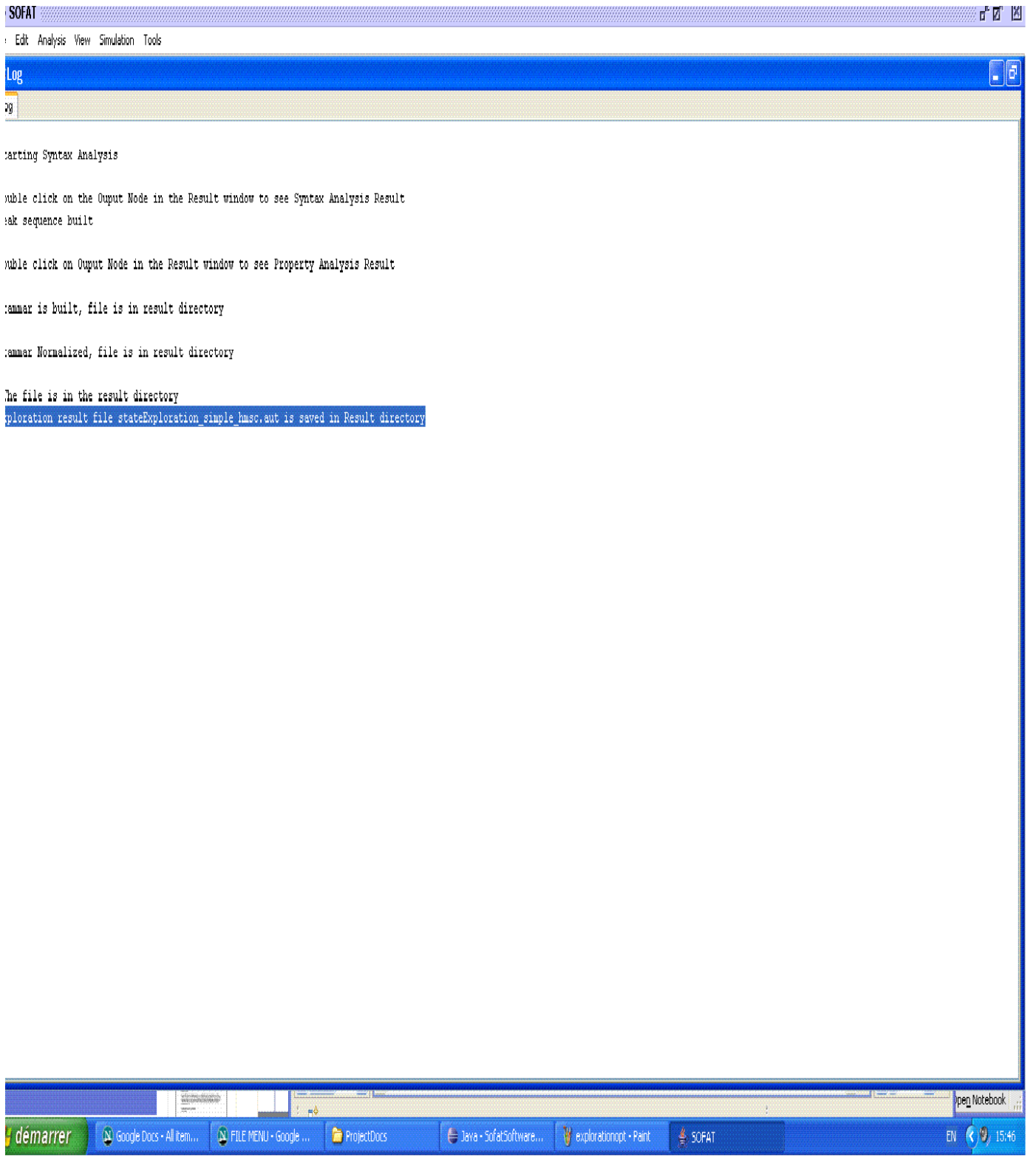
Figure 28: Use of Exploration option

Figure 29: Exploration Log Window

Grammar Unfolding.

The exploration item of the simulation menu builds (a part of) the interleaved semantics of an MSC description. Grammar unfolding builds the partial order semantics of the same description under the form of an evnt structure. Again, this representation can be infinite, and this unfolding is performed up to a certain depth, i.e. in this case up to a certain number of rules rewritings.

The user can either view the Grammar Unfolding by selecting the display option or save it in a dot format. The user has to select the depth from the drop down list in the frame that is displayed when the option is selected. The file is saved in the Result directory. The file name is of the form name of msc+"_GrammarUnfolding".dot. Figure 30 shows the use of this option. The user has selected to unfold the grammar up to depth 5. Also, the user has decided to display the grammar instead of saving it as a dot file. The unfolding is displayed as shown in the Figure 31.
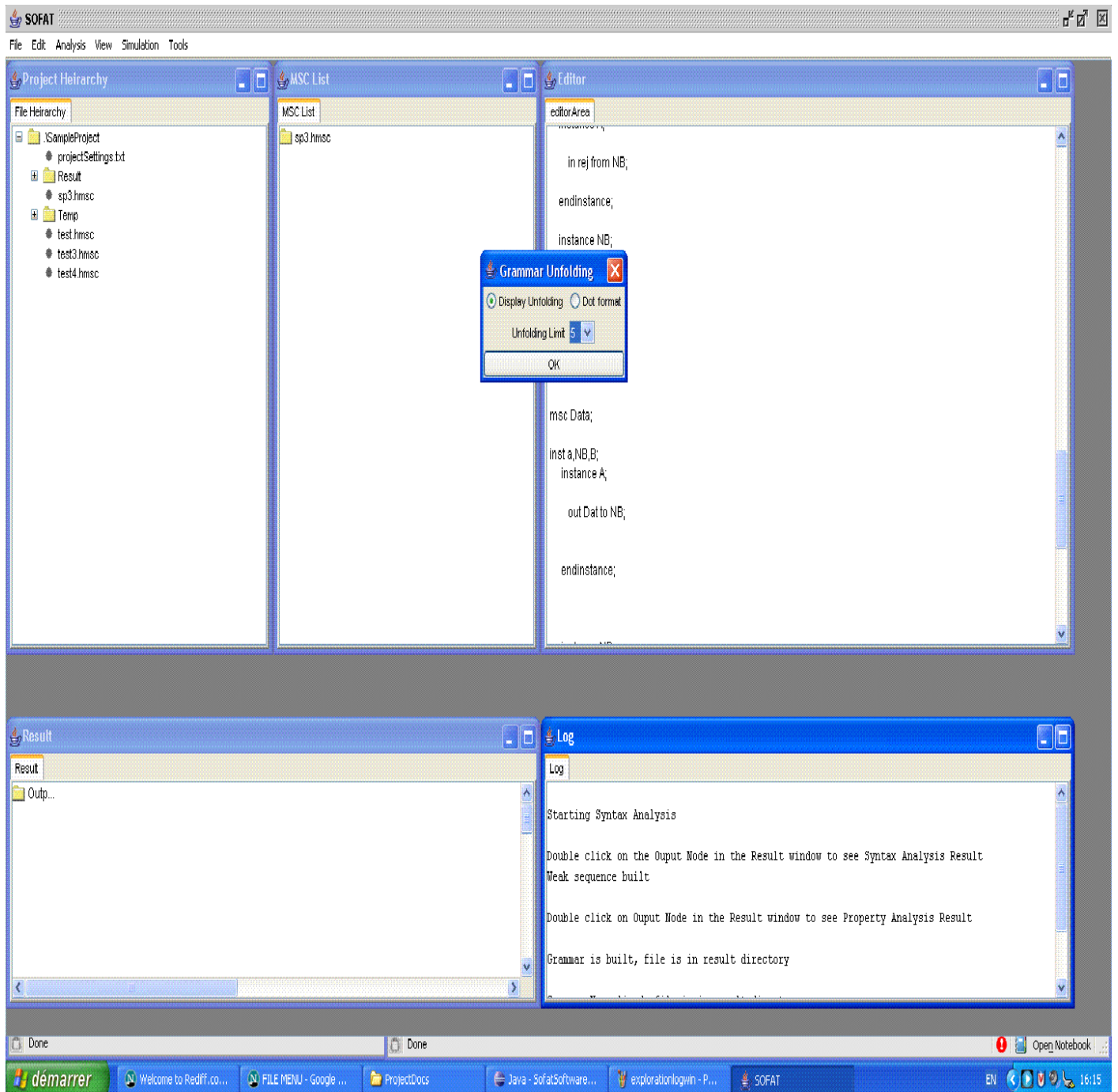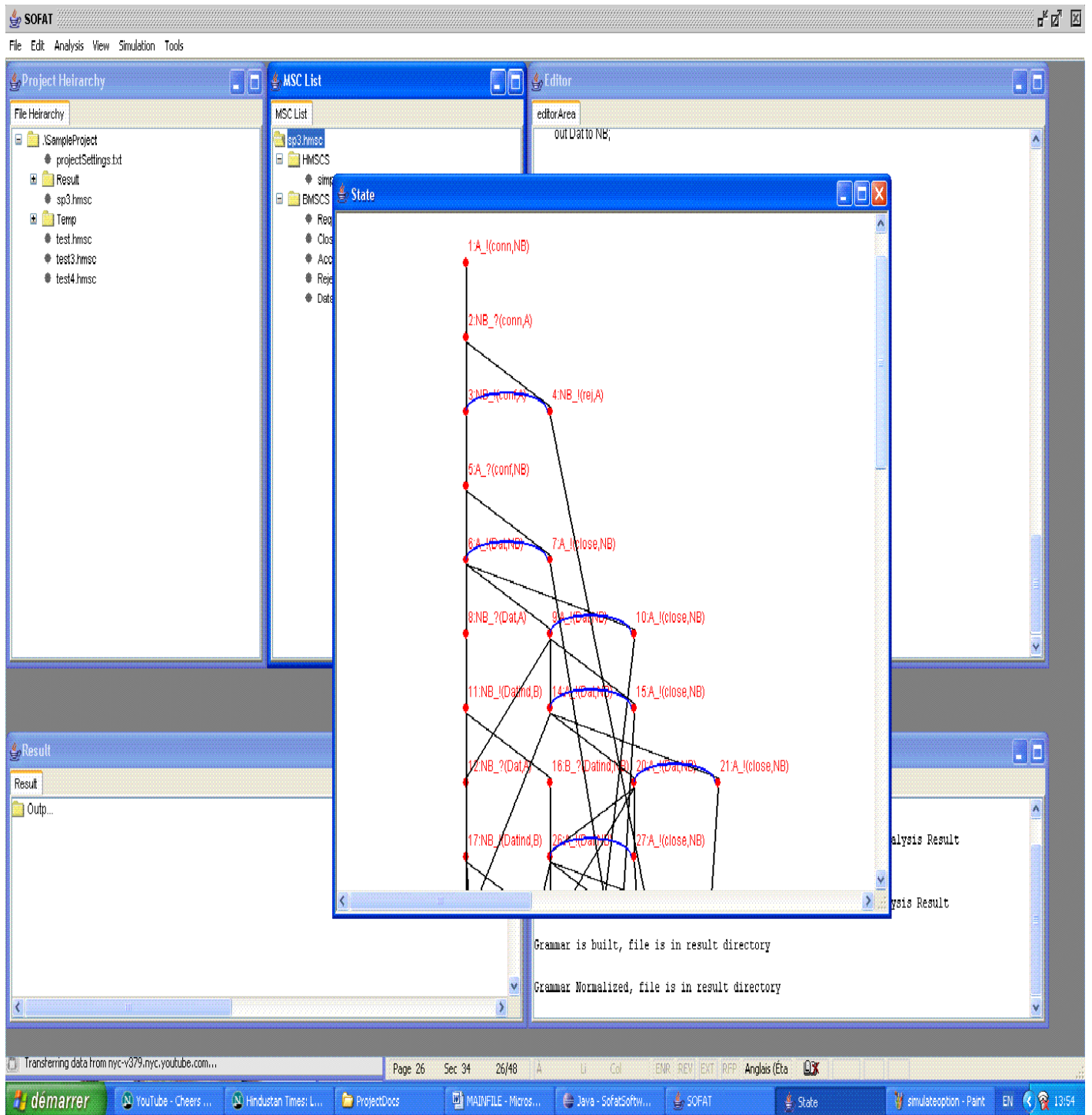
Figure 30: Grammar Unfolding

Figure 31: Displaying a grammar unfolding

3.6.6 TOOLS MENU

The tools menu gathers miscellaneous functions that could not be classified elsewhere, to manipulate MSCs or to customize SOFAT's use. The following items are available to the user:

Preferences.

The user can set his prefences for the following options:

Font Size. This is the size of the font used by the editor window. Default size is set to 12.

Font Type - Plain, Bold and Italic. Default is Plain.

Graphical Output Type – Format used to display MSCs. Post Script and GIF. Default is PostScript.

Once the user has set the Preferences, he/she will not have to do it again. When SOFAT is closed, existing Preferences are saved in a file called "Preferences.txt". The next time the user tries to load the software the preferences are read from the preference file. The screen shot of the preferences option can be seen in figure 32.
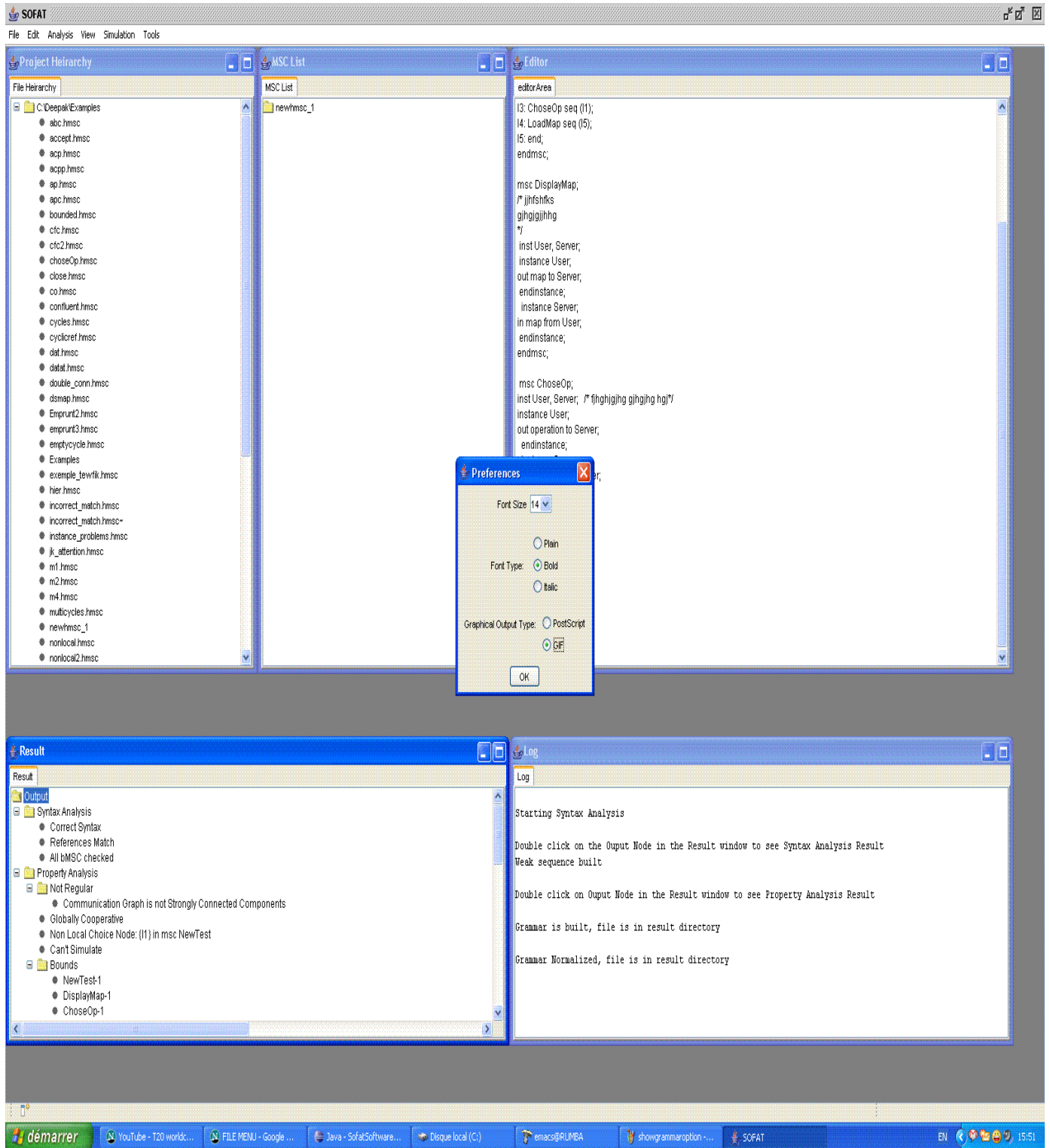
Figure 32: Preferences Window

3.6.6.1 Show Grammar.

This item is used to display the grammar rules in separate windows. Every window contains a graphical representation of the rules of the grammar. This functionality is mainly used for debugging purposes. In the example of Figure 33, rules Rule 1, Rule 2, Rule 3, Rule 4, Rule 5 and Rule 6 are displayed in separate windows.
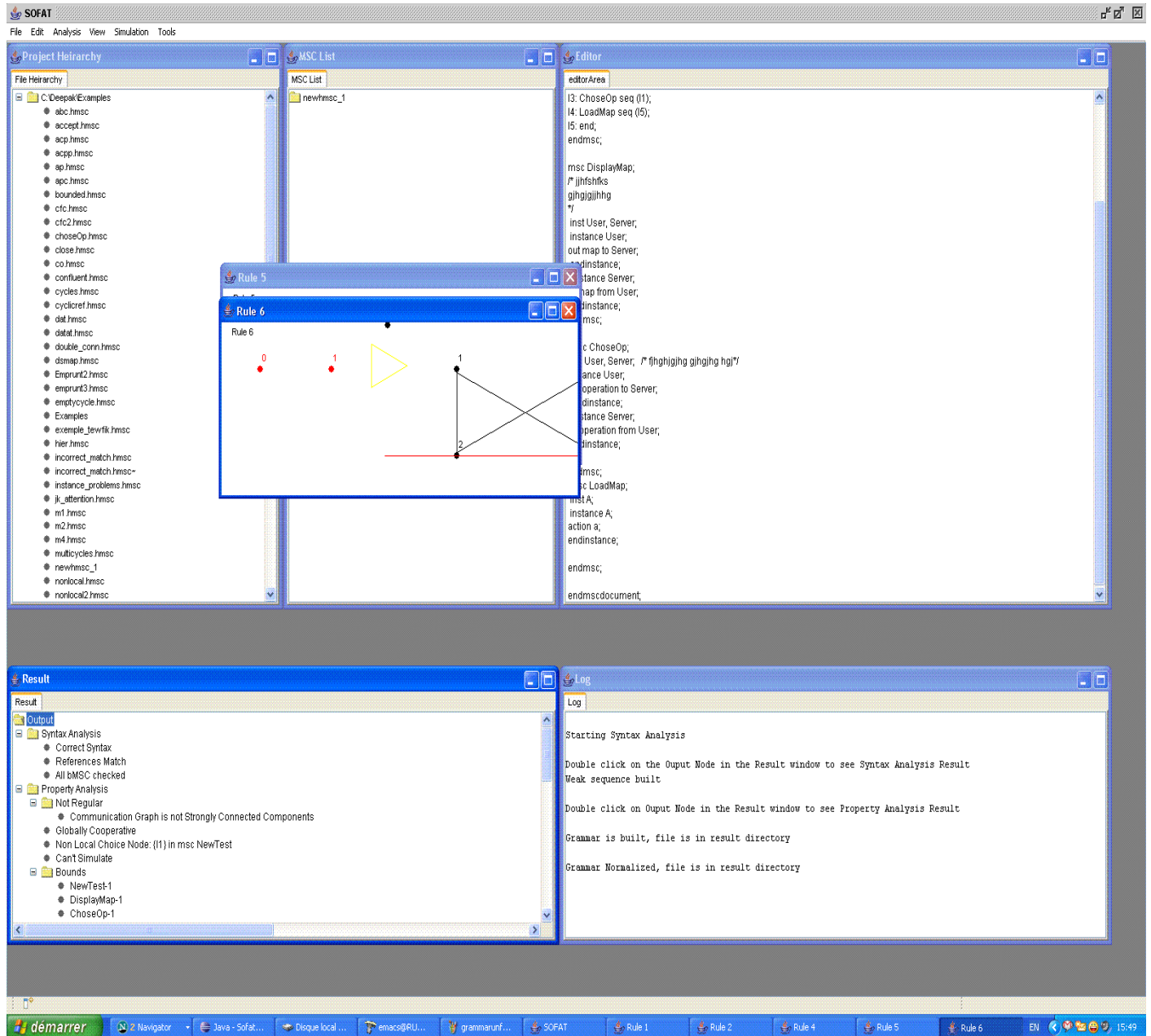

Figure 33: Show grammar option

.

3.6.6.2 Unfolding (MSC).

Unfolding a HMSC consists in generating all path that start from the initial node of the HMSC up to a certain number of bMSC. When the unfolding item is chosen in the tool menu, a popup window appears. The user has to select a depth for the unfolding (see Figure 34). The result of the unfolding is displayed in a new window, and saved either in a Post Script format or GIF format depending on the user preferences. In Figure 34, he user has selected depth 4. The Log window displays the name of the resulting file and the directory where it is saved. In this case it is   "C:\Deepak\Examples\Result\Unfolding_NewTest_4.ps" is saved in the Result directory of the project.
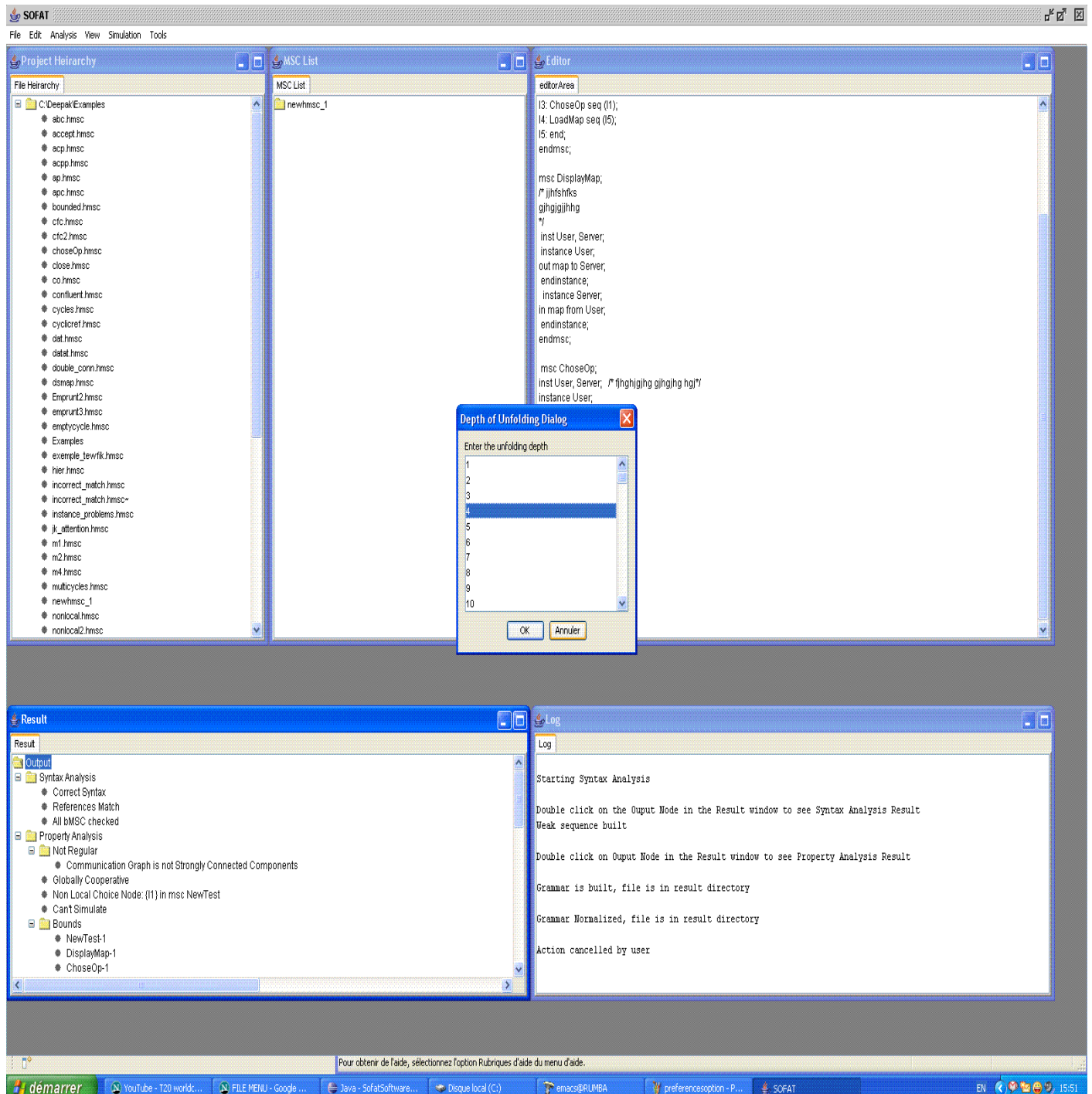

Figure 34: MSC Unfolding option

3.6.6.3 Highlighting

To simplify the edition and the correction of errors in MSC specifications, SOFAT can highlight MSC reserved keywords. Highlighting shows reserved keywords with different colors. MSC reserved keywords are highlighted in blue, and comments in red. Figure 35 shows the highlighting of a text within the editor window. When a file is saved, highlighting disappears, and have to be re-activated. Highlighting is not perfomed online during the edition of an MSC description for performance reasons.

Figure 35:Highlighting Keywords

3.7 Example

To illustrate the main functionalities of SOFAT V3, we propose a complete example. The chosen example is a simple data transfer protocol, involving three instances : A, NB and B. Instance A must transfer data packets to B via a frontend NB. Before transmitting an arbitrary number of data packets, A has to establish a connection with NB. Data packets are then simply transmitted from A to NB, which forwards them to B. When A has no more data packets to sent, it closes the session

```
mscdocument simple_protocol;

msc simple_hmsc ;

expr L1;

  L1: Request seq  (L2);
  L2 : connect seq (L3 alt L4);
  L3: Accept seq (L6) ;
  L4: Reject seq (L5) ;
  L5: end;
  L6: connect seq ( L7 alt L8);
  L7: Data seq (L6);
  L8: Close seq (L9);
  L9: end;
endmsc;


msc Request;
inst A,NB ;

   instance A;

     out conn to NB;

   endinstance;

   instance NB;

    in conn from A;

   endinstance;

endmsc;
```

```
msc Close;
inst A,NB ;

   instance A;

     out close to NB;

   endinstance;

   instance NB;

    in close from A;

   endinstance;

endmsc;

msc Accept;
inst A,NB ;

   instance A;

     in conf from NB;

   endinstance;

   instance NB;

    out conf to A;

   endinstance;

endmsc;
```

```
msc Reject;
inst A,NB ;

   instance A;

     in rej from NB;

   endinstance;

   instance NB;

    out rej to A;

   endinstance;

endmsc;


msc Data;

inst A,NB,B;


   instance A;

     out Dat to NB;

   endinstance;

   instance NB;

     in Dat from A;
     out Datind to B;

   endinstance;

   instance B;

     in Datind from NB;

   endinstance;

endmsc;

endmscdocument;
```

The simple protocol example.

The various operations described in the software document are performed on the above example and the output obtained is shown in screen captures. To run all manipulations presented in this section, you can download the simple protocol exemple contained int the EXAMPLE.ZIP archive available from SOFAT's webpage.

First of all let us create a new project, using the file menu, item new > project.
Chose a project name (TestProject, for instance- we will use this name in the sequel)
When the project is created, chose item add MSC in the file menu. This opens a new file exploration popup window, from which you can select the sp.hmsc file from the directory where you have uncompressed the EXAMPLE.zip archive.

The projet window now conatins a Hierarchy composed of a project TestProject, two directiories result and temp, and a MSC description sp.hmsc.

When the sp.hmsc has been added to the project, select it, and run the syntactic analysis of the analysis window. Once the syntactic analysis is complete (file sp.hmsc is syntactically correct) you can display all MSCs using the View menu. The result of syntax analysis is displayed in Figure 36.

Figure 37 shows the graphical representation of HMSC simple_hmsc in the description above. To display this HMSC, use the view menu, select High level MSCs, and select MSC simple_hmsc from the windows that pops up. Figure 38 shows a BMSC produced by SOFAT (chosing this time the item BMSC of the view menu).
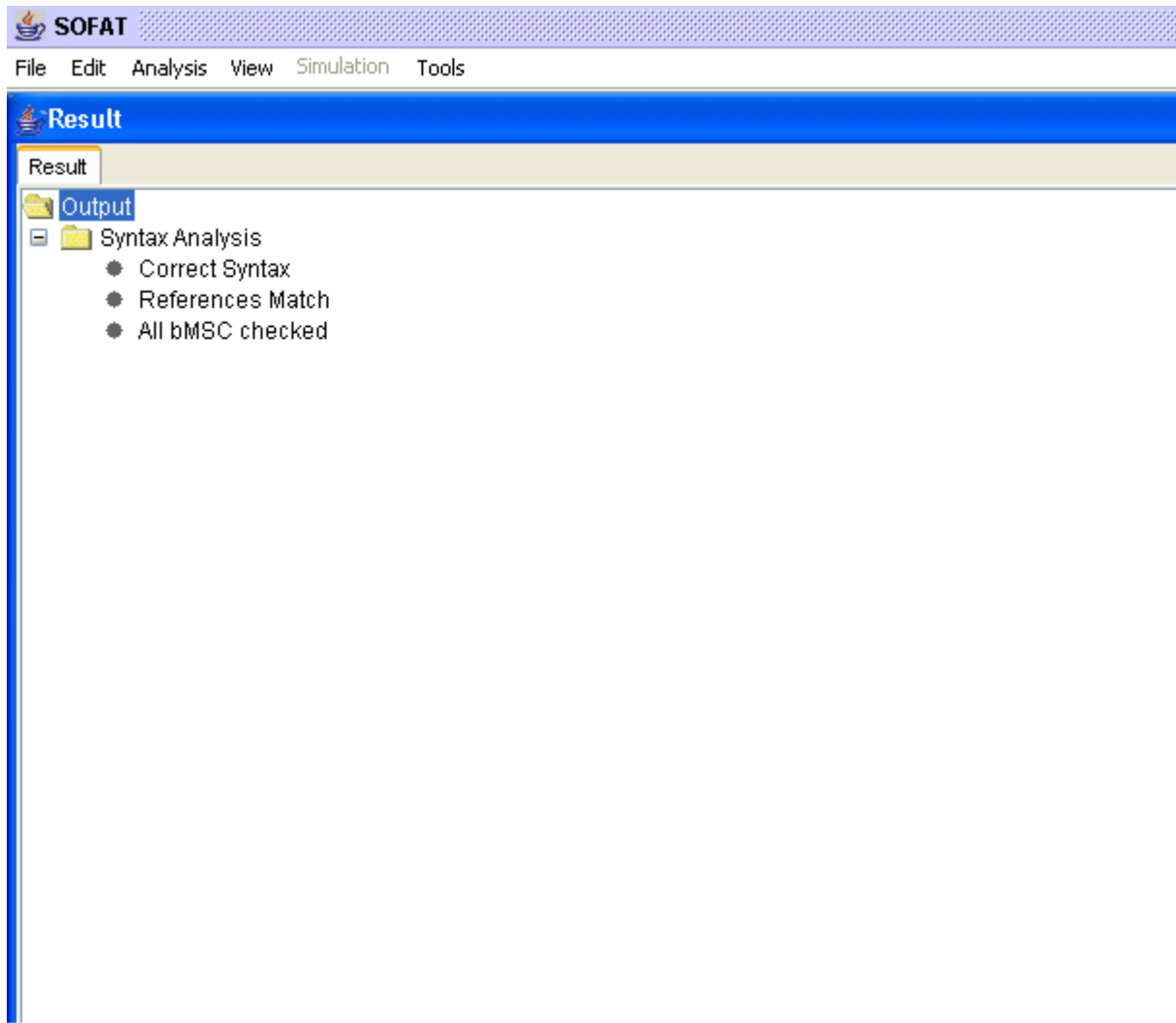
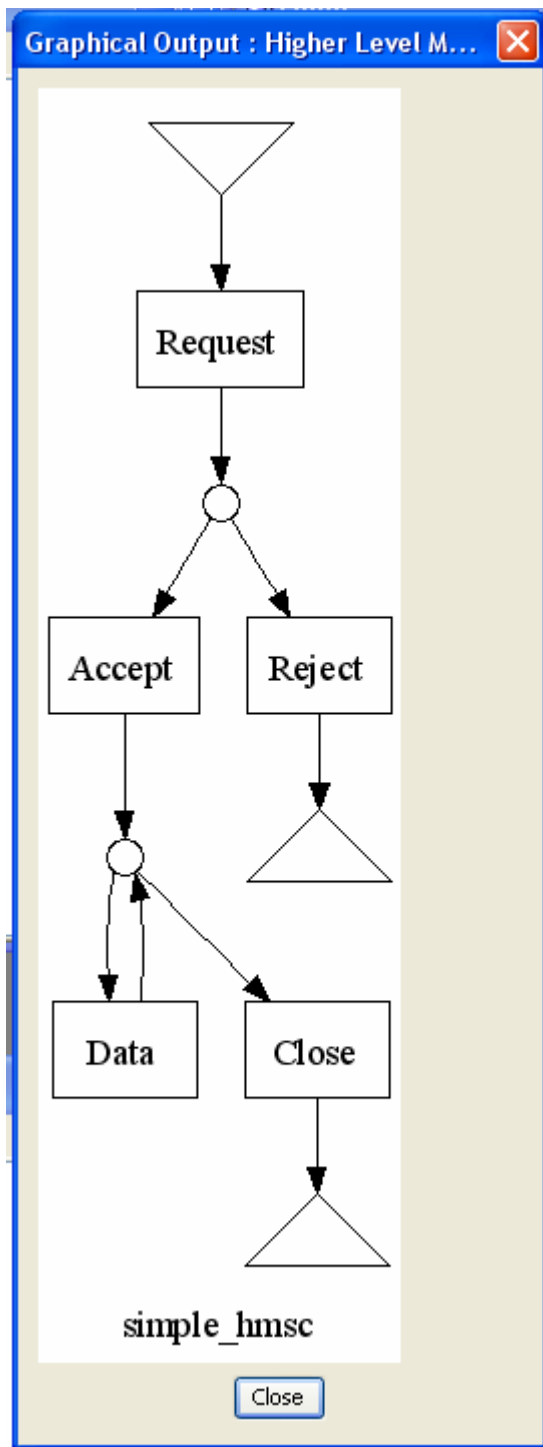**Fig 36: Syntax Analysis Output for Example 1.**

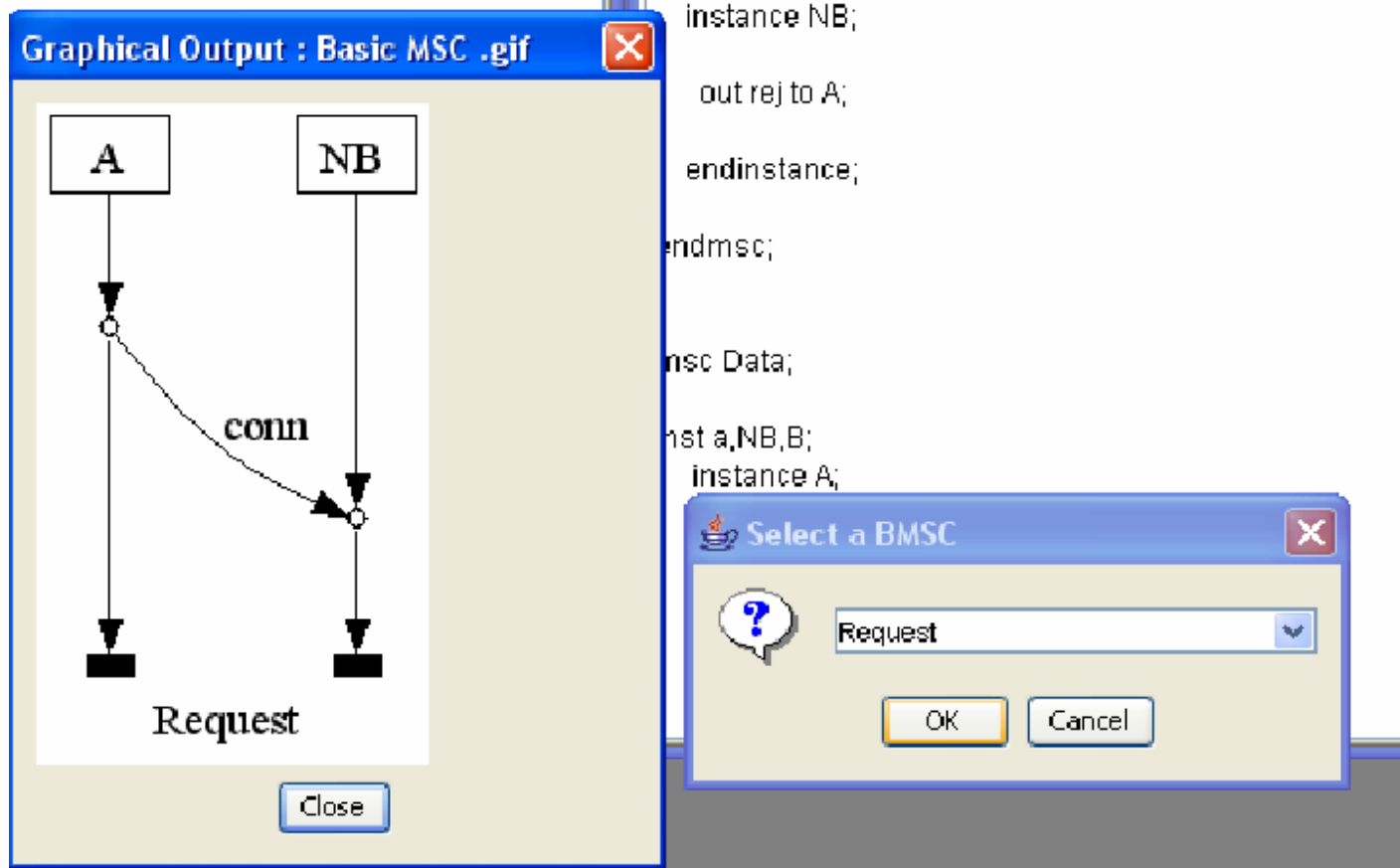**Fig 37: Graphical View of HMSC simple_hmsc in the simple Protocol example.**

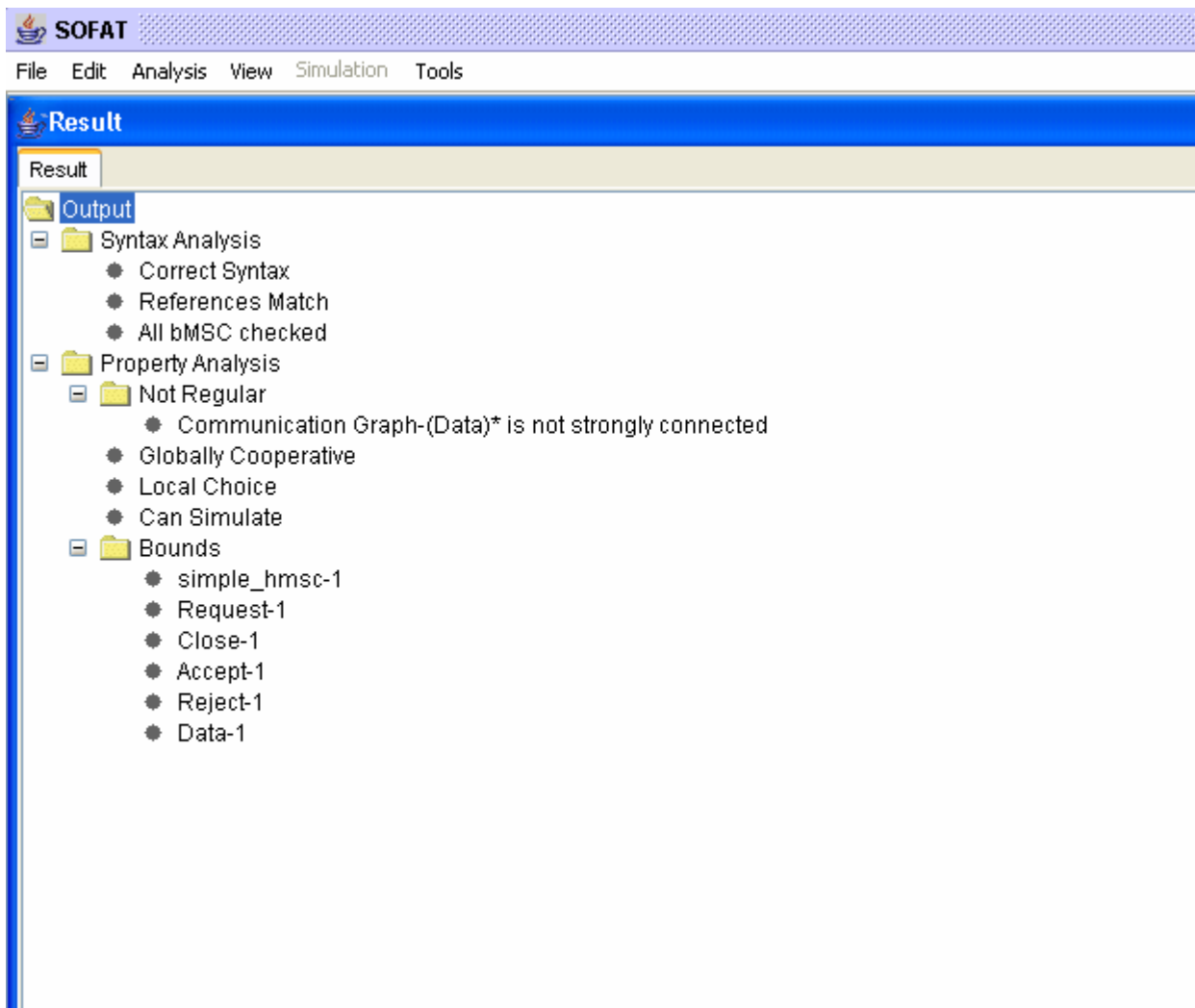**Figure 38: Graphical view of bMSC Request.**

Fig 39: Property Analysis.

Now let us analyze the properties of this MSC description, with the analysis menu, item property analysis.

As can be seen from the results above the Example MSC is not Regular. The simple_hmsc conatins a cycle that uses Data BMSC. Is this MSC, messages can be sent without acknowledgement, and then the description has an infinite state space. Instance A can overflow the system with Data messages. Note however that this description is globally Cooperative. Since the MSC is simulable it is possible to perform simulation on it. The bound value of 1 indicates that all masic MSCs comport a run in which the contents of communication channels do notexceed one transiting message. The HMSC is local choice since the choice between BMSC Accept and Reject is decided by instance NB while the choice between Data and Close is decided by instance A. The result of property analysis is shown in Figure 39.

```
SOFAT
File  Edit  Analysis  View  Simulation  Tools

Editor
editorArea

mscdocument Accept_document ;

msc Split_Accept ;
expr L1;
  L1: Accept_0 seq (L2);
  L2: end;
endmsc;
msc Accept_0;
inst A,NB;
  instance A;
    in conf from NB;
  endinstance;
  instance NB;
    out conf to A;
  endinstance;
endmsc;
endmscdocument;
```

Figure 41: Output of split MSC (only BMSC) performed on bMSC Accept.

Let us split bMSC Accept into atoms. For this, we select basic MSC Accept from the MSC list tab/window, right click on it. A menu appears. Chose split MSC from this menu. Then choose the first option to create a separate file just for the Accept BMSC. The result of this split is the HMSC shown in Figure 41.

```
SOFAT
File  Edit  Analysis  View  Simulation  Tools

Editor

editorArea

mscdocument simple_protocol;

msc simple_hmsc ;

expr L1;

 L1: Request seq  (L2);
 L2 : connect seq (L3 alt L4);
 L3: Split_Accept seq (L6) ;
 L4: Reject seq (L5) ;
 L5: end;
 L6: connect seq ( L7 alt L8);
 L7: Data seq (L6);
 L8: Close seq (L9);
 L9: end;
endmsc;


msc Split_Accept ;
expr L1;
 L1: Accept_0 seq (L2);
 L2: end;
endmsc;
msc Accept_0;
inst A,NB;
 instance A;
  in conf from NB;
 endinstance;
 instance NB;
  out conf to A;
 endinstance;
endmsc;
```

Figure 42 Output of split MSC (New HMSC and Overwrite HMSC option) performed on bMSC Accept.

The other possibility when a bMSC is split is to save the new description in a different file. In this case the operation is same as described previously except choose the second option of creating a separate file for the entire MSC using the split MSC or third option of overwriting the existing HMSC file. The result is shown in Figure 42.

```
SOFAT
File  Edit  Analysis  View  Simulation  Tools

Editor
 editorArea
mscdocument Data_document;
msc Data;

inst a,NB,B;
    instance A;

        out Dat to NB;

    endinstance;



    instance NB;

      in Dat from A;
      out Datind to B;

    endinstance;

    instance B;

      in Datind from NB;

    endinstance;

endmsc;
endmscdocument;
```

Figure 43: Output of Create Separate MSC File performed on BMSC Data.

The last possibility when splitting is to create a separate file for the split bMSC only.For this, choose the option of creating a separate file for the BMSC selected from the MSC List tab/window after right clicking on it. The result is shown in Figure 43.
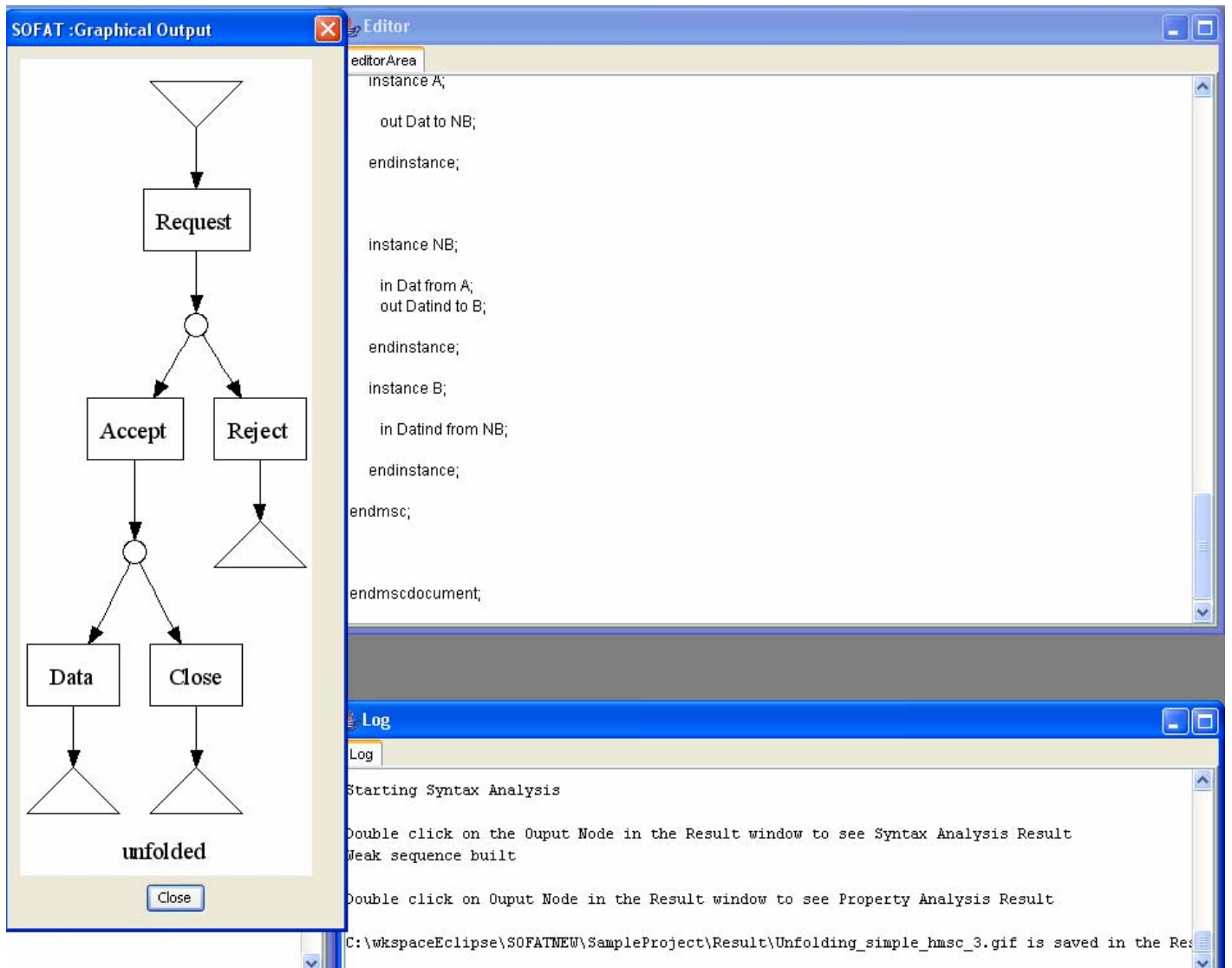
Figure 44: Output of MSC Unfolding performed with depth 3.

The simple_hmsc is unfolded to a depth of three by clicking on the MSC Unfold option in the Tools Menu and selecting the depth from the drop down list. Figure 44 shows the unfoldin of HMSC simple_hmsc to a depth of 3.

3.8 Organization of the software

SOFAT is distributed as a jar file named SOFAT.jar, located in the lib directory situated in the installation directory. If you have installed SOFAT V3 in directory C:\Progam Files\SOFATV3, then this archive should be located at C:\Progam Files\SOFATV3\lib\sofat.jar.

If you have installed a java Virtual Machine on your computer, a double click on the archive should launch SOFAT.

3.8.1 Directories

SOFAT is launched in the lib directory. All newly created projects appear as subdirectories of the lib directory. Each project directory contains the msc descriptions used within the project (these files are usually named with the extension ".hmsc") and two directories used to store the results of the analyses and the temporary files created by SOFAT.

Result - The directory is created within the project directory. All files produced during the working of the software are stored in the directory. The files stores in this directory are the grammar files (for simulation), the graphical output files, etc.

Temp - The directory is created in the project directory. It is used as buffer while performing certain operations and no files are stored in it on a permanent basis.

3.8.2 Files

Sofat.jar : SOFAT executable.

Preference.txt - This file is created the first after the first execution of SOFAT. It contains all the user preferences, the name of the last working file and directory. When is executed another time, the preferences are read from this file, and the user can continue from the state in wich the software was when shut down.

In addition to these generic files located in the lib directory of the installation, each project contains a file named projectSettings.txt . It contains all information related to the current project (hmsc documents, etc). These projectSettings.txt files **should not be deleted**.

3.9 Required Software

SOFAT has been developed entirely in JAVA. It then needs a Java virtual machine to run. It has been mainly tested on Windows machines, but works on MAC, SOLARIS, or LINUX machines.

Java - The software is tested with JRE 1.4 and higher.

Dot- It is required for generating the graphical output for the Message Sequence Charts.

Output formats

.gram is used for the grammar files that are generated.

.dot - The graphical output generated for the Message Sequence Charts are saved in dot format.

.aut - The automaton file generated for grammar unfolding.

3.10 Support

Sofat is provided on an "as is" basis. The development team claims no responsibility for consequences of bugs in the program. However, SOFAT rely on formal bases, and on a standardized language. That is all algorithms implemented in SOFAT have been proved correct. If you find a bug in SOFAT, it is certainly correctible. Please contact the sofat developers, we will do our best to fix your problem.

Similarly, SOFAT follows the state of the art on scenario theory. New releases are frequently published on the tool's webpage :

www.irisa.fr\distribcom\Prototypes\SOFAT\index.html

If you feel that some functionality should be implemented, feel free to contact us, and ask for it in the next release of the tool.

CONTACT : sofat@irisa.fr

4 Bibliography

[Alur05] Rajeev Alur, Kousha Etessami, Mihalis Yannakakis: Realizability and verification of MSC graphs. Theor. Comput. Sci. 331(1): 97-114 (2005)

[Clarke99] E.M.Clarke, O. Grumberg, D. Peled: ``Model Checking'', MIT Press, December 1999.

[Khendek99] M.M. Abdalla, F. Khendek and G. Butler, "New Results on Deriving SDL Specifications from MSCs", in the Proceedings of SDL Forum'99, Elsevier Science B. V., R. Dssouli, G.v. Bochmann and Y. Lahav (eds.), Montreal, Canada, June 21-25, 1999.

[Genest02] Blaise Genest, Anca Muscholl, Helmut Seidl, Marc Zeitoun: Infinite-State High-Level MSCs: Model-Checking and Realizability. ICALP 2002: 657-668

[Genest04] Blaise Genest, Dietrich Kuske, Anca Muscholl: A Kleene Theorem for a Class of Communicating Automata with Effective Algorithms. DLT 2004, 30-48, LNCS 3340.

[Helouet99] Loïc Hélouët, A Simulation Framework for Message Sequence Charts, 9th SDL Forum, 21-25 June 1999, Montréal.

[Helouet00] Loïc Hélouët, Claude Jard, Conditions for synthesis of communicating automata from HMSCs, 5th International Workshop on Formal Methods for Industrial Critical Systems (FMICS), Berlin, 3-4 april 2000.

[Holzmann99] Gerard J. Holzmann and Margaret H. Smith, Software Model Checking, Proceedings of FORTE 1999, FORTE}", pages 481-497, 1999.

[Reniers99] S. Mauw and M.A. Reniers. Operational Semantics for MSC'96, Computer Networks and ISDN Systems 31(17):1785-1799, 1999. Elsevier Science B.V.

[Z120] ITU-T Recommendation Z.120 : Message Sequence Charts.

[Z120AnnexB]. Recommendation Z.120 annex B: Algebraic Semantics of Message Sequence Charts, ITU-TS, Geneva, 1998